VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

# Dynamické generování XML formulářů na základě XSD

# Dynamic XML Forms Generation From XSD Schema

2012 Roman Ollé

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

# Zadání bakalářské práce

Student: **Roman Ollé**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Dynamické generování XML formulářů na základě XSD
Dynamic XML Forms Generation From XSD Schema

Zásady pro vypracování:

Postup pro vypracování (specifické cíle práce):

1. Prostudujte podrobně problematiku XML schémat a možnosti generování XML formulářů.
2. Navrhněte desktopovou aplikaci podporující dynamické generování XML formulářů ze zadaného XSD souboru a jejich výsledné uložení do XML souboru.
3. Detailně zpracujte specifikaci a samotný návrh řešení.
4. Uvedený návrh naimplementujte pomocí technologie Java Swing a výslednou aplikaci otestujte.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Oldřich Vašut**

Datum zadání: 18.11.2011

Datum odevzdání: 04.05.2012


doc. Dr. Ing. Eduard Sojka
*vedoucí katedry*

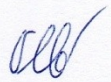
prof. RNDr. Václav Snášel, CSc.
*děkan fakulty*

# Prohlášení studenta

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Ve Studénce dne 26.5.2012



Roman Ollé

# Poděkování

Rád bych poděkoval svému vedoucímu mojí bakalářské práce panu Ing. Oldřichu Vašutovi za čas pro moje dotazy, odbornou pomoc a konzultace při vytváření této práce.

# Abstrakt

Tato bakalářská práce se zabývá navržením a naimplementováním aplikace pro dynamické vygenerování formulářů podle XSD dokumentů a jejich následnému uložení do XML souboru. V práci jsou podrobně popsány možností jednotlivých formátů, jejich možnosti a vlastnosti, jež aplikace využívá. Dále jsou popsány i technologie a jejích částí použité pro tvorbu aplikace. Do této kategorie patří programovací jazyk Java a její knihovny Swing a SAX parser. Pro aplikaci jsou zde podrobně rozebrány nejdůležitější a nejsložitější části v analýze řešení a jejich řešení při implementaci. Konec práce obsahuje závěrečné shrnutí a popis nedostatků a možné vylepšení aplikace.

# Klíčová slova

Generování formulářů, vytváření XML, Java SE, XML, XML schéma, formuláře, Swing, Sax parser

# Abstract

This Bachelor thesis is concerned with suggestion and implementation for application for dynamic generating for forms according to XSD documents and their subsequent input to the XML file. Formats are described in detail. Options of formats and attribute of formats are described as well. The technology and parts of this technology are used for creation of application. Java programming language and its libraries Swing and SAX parser are included in this category. The most important and difficult parts of solution in the analysis are described in detail in the application and their solution implementation. The end of the work includes a final summary and description of shortages and possible improvement of the applications.

# Key words

Generating forms, create XMLs, Java SE, XML, XML schema, forms, Swing, Sax parser

# Seznam použitých symbolů a zkratek

| | |
|---|---|
| API | Application Programming Interface |
| AWT | Abstract Window Toolkit |
| DOM | Document Object Model |
| DTD | Document Type Declaration |
| GUI | Graphical User Interface |
| HTML | HyperText Markup Language |
| IDE | Integrated Development Environment |
| ISBN | International Standard Book Number |
| ISO | International Organization for Standardization |
| JAR | Java Archive |
| Java SE | Java Standard Edition |
| JVM | Java Virtual Machine |
| OASIS | Organization for the Advancement of Structured Information Standards |
| PDF | Portable Document Format |
| PSVI | Post-Schema Validation Infoset |
| RELAX NG | REgular LAnguage for XML Next Generation |
| RSS | Really Simple Syndication |
| SAX | Simple API for XML |
| SGML | Standard Generalized Markup Language |
| UML | Unified Modeling Languagec |
| URL | Uniform Resource Locator |
| VP | Visual Basic |
| W3C | World Wide Web Consortium |
| XDR | XML-Data Reduced |
| XHTML | eXtensible HyperText Markup Language |

| | |
|---|---|
| XML | eXtensible Markup Language |
| XSD | XML Schema Definition |

# Seznam obrázků

# Seznam příkladů

# Seznam tabulek

# Obsah

# 1 Úvod

V dnešní době jsou XML [1] soubory stále častějším úložištěm pro data. A to díky jednoduché syntaxi a návrhu. Velkou výhodou je její multiplatformní využití. Navíc XML dokumenty jsou k dispozici bez licence, takže pro vlastní využití se neplatí. XML dokumenty slouží jako úložiště pro různá data a informace.

Lze použít pro výměnu informací mezi obchodními partnery v elektronické podobě, díky jeho jednoduchosti. Tím jsou myšleny třeba objednávky, faktury a tak podobně. Další možností je použití pro publikování dat, neboť data uložena v XML můžeme podle potřeby pomocí stylů převést do mnoha různých formátů jako jsou PDF, webové stránky atp. XML dokumenty mohou být použity i jako konfigurační soubory pro aplikace. A to tak, že dokument obsahuje různé parametry s nastavením, které aplikace používá. Tímto způsobem nahradí konfigurační soubory INI, systémové registry a další složité dokumenty. Pomocí XML byl skriptovací jazyk HTML nahrazený novým XHTML. XHTML používá z HTML většinu značek, ale celková syntaxe byla pozměněna, aby vyhovovala standardu XML., tudíž webové stránky jsou v podstatě XML dokumenty.

XML dokumenty lze vytvářet několika způsoby, prvním základním je vytváření nových dokumentů ručně v nějakém textovém editoru, jenomže tento způsob je extrémně náročný, zdlouhavý, složitý, zabere mnoho času a lze v něm jednoduše udělat chybu jako je křížení elementů, neukončení elementů, apod., což může rozhodit celou aplikaci, která tento soubor použije.

Další možností by šlo udělat statické programové uložení, kde se vytvoří pro každý nový typ XML nový program, jenž bude na základě zdrojového kódu ukládat nová data se stejnou kostrou, tento typ vkládaní je už poměrně dobrý, ale pořád si musí člověk vytvářet nový projekt na generování a sestavení každého typu XML dokumentu zvlášť.

Dalším způsobem je vytvořit aplikaci, která na základě jazyka pro specifikaci XML souboru, který vlastně definuje, jak bude výsledný XML dokumentu vypadat, tzn. vytvoří automaticky strom XML souboru podle zadaného schématu. Velkou výhodou je, že výsledné dokumenty vytvořené podle jazyka pro specifikaci XML souboru budou mít stejnou strukturu, což zamezuje chybám ve výsledných souborech XML. Z toho vyplývá, že parsery pohodlně zpracují každý XML soubor, které budou vytvořeny podle souboru pro specifikaci, aniž by nastal nějaký problém ve zpracování. Pro celý tento bude vytvořená aplikace, jenž nabídne formulář, odpovídající libovolnému XML schématu, pro vložení hodnot do elementů a atributů a později tyto data uloží do XML souboru, tudíž všechny XML souboru budou mít stejnou strukturu.

Cílem této bakalářské práce bude vytvoření právě výše zmíněné aplikace pro vygenerování a ukládaní libovolných XML dokumentů, dle nadefinovaného XML schéma. Máme několik druhů dokumentů obsahující XML schémata a plno z těchto dokumentu nemá tolik možností pro

vytvoření libovolného schématu, asi tím s nejvíce možnostmi, mnohem kvalitnější a propracovanější je technologie s názvem XSD, a proto je tato technologii XSD ideální pro tuto práci. Celá aplikace bude naprogramována v programovací jazyk Java SE, v němž vytvořím desktopovou aplikaci, která po vložení libovolného XML schématu vygeneruje formulář odpovídající tomuto schématu, do nějž se následně budou vkládat data, jenž se uloží do již zmíněného XML dokumentu.

Celou tuto bakalářskou práci bych rozdělil na 3 části, podle práci s jednotlivými soubory, které se budou zrovna zapotřebí. Těchto části se při práci budu držet a patřičně takto rozdělím i samotný program.

První části je načtení souboru s XML schématem, který uživatel načte do aplikace z disku počítače, toto schéma musí být validní. Vložené schéma následně rozparsuju pomocí parseru a následně vytvořím strom s jednotlivými elementy a údaji o jejich podobě, Tento strom budu představovat složené schéma jednotlivých částí vloženého souboru, které bude dost připomínat, jak by měl vypadat výchozí formulář pro vkládání dat. Z parserů mám na výběr z několika možností, nejpoužívanější jsou asi DOM a SAX parser.

Druhou částí je dle předchozího odstavce vygenerování formuláře, který odpovídá vloženému XML schématu v první části práce. Samotný formulář bude vytvořen ve Swingu, což je GUI widgetu Javy, tento vygenerovaný formulář bude dynamicky se měnící, dle akce uživatele, jako je přidání a odebíraní elementů, vybíraní z elementů, posouvání elementů nahoru a dolů, atp., ale tyto operace musí být nadefinovány ve vloženém XML schématu. Při těchto operacích vytvořím nový strom s elementy a nastavením formuláře.

Poslední částí bude uložení dat z formuláře. Tato část aplikace projde strom s elementy a vybere z formulářů data, která musí být odpovídat správnému tvaru podle údajů z vloženého schématu a následně tyto data uloží do XML dokumentu, jehož stromová kostra bude odpovídat nadefinované kostře v XML schématu. Touto událostí se ukončí práce s vloženým XML schématem s možností začít celý nový proces dalšího XML schématu.

# 2 Technologie

## 2.1 XML dokument

XML je „rozšířitelný značkovací jazyk“ a pochází z oblasti zaměřené na uchování a zpracování textových dokumentů. Vychází z jazyka SGML. XML je standardem W3C a v dnešní době je jeden z hlavních formátů na výměnu dat strukturovaným způsobem.

XML popisuje data nezávislé na platformě, jak na typu počítače, tak na operačním systému, což zaručuje přenositelnost dat ba dokonce celých programů pracující s těmito daty. To je dokázáno tím, že obsah dokumentů jsou v podstatě textové informace a navíc můžeme u nich nastavit kódování znaků v hlavičce XML dokumentu.

Dokument XML je jednoduchý otevřený formát, to je, že můžeme libovolně dle potřeby doplňovat značky. Značky můžou mít libovolný pojmenování, tento název musí dodržovat pár základních omezení. Každý název značky by měl jasně určovat význam uložených dat. Značka nese pouze informace, nikoliv formátování.

Hlavním prvkem XML dokumentu je element. Představuje počáteční a koncovou značku a informaci uloženou mezi nimi. Každý dokument musí obsahovat právě jeden kořenový element, který obaluje zbývající elementy.

Dalším důležitým prvkem je atribut, který je umístěn do počáteční značky elementu a značí se dvojicí údajů název="hodnota", hodnoty musí být uzavřeny do horních uvozovek.

Dokument má dvě hlavní oblasti použití, prvním jsou tzv. datově orientované dokumenty, jejichž hlavním úkolem je přenos dat mezi aplikacemi, dá se říct, že je to úložiště dat. Struktura má plno značek obalující informace. Lze využít jak pro uložení dat pro aplikaci, tak například i pro konfigurační soubory aplikace, jenž jsou v dnešní době velmi častě používané. Tento typ kódu je zobrazen v Příklad č.1, jenž kořenový element tvoří element produkty, který obsahuje produkty s atributem id a s elementy cena a název s hodnotou. [2]

```
<produkty>
 <produkt id="369">
   <nazev>Čokoláda</nazev>
   <cena>14.50</cena>
 </produkt>
 <produkt id="571">
   <nazev>Jogurt</nazev>
```

```
    <cena>6.70</cena>
  </produkt>
</produkty>
```

Příklad č.1 Ukázka jednoduchého datového XML kódu

Druhou oblast tvoří dokumenty orientované na sdělení, kde je už méně značek, tyto značky buďto obalují nebo jsou vkládány do vět. Tímto způsobem se připravují knihy, WWW stránky a jiné počítačové dokumenty. Tento typ dokumentu slouží především pro formátování textů, do výsledné podoby použitelné pro uživatele se musí použít další technologie, aby tento XML dokument zpracovala. Příklad č.2 zobrazuje odstavec v HTML s textem v kterém jsou elementy strong a em, jenž mají představovat tučný text a italiku.

```
<div>Nějaky odstavec s <strong>tučným text</strong>, text
pokračuje <em>kurzívou</em> a pak odstavce skončí.</div>
```

Příklad č.2 Dokument orientovaný na sdělení

# 2.2 XML schémata

Hlavním významem XML schémat je definování výsledného souboru s nějakým značkovacím jazykem založeným na XML. Výhodou výsledného souboru je stromová struktura odpovídající schématu, tudíž zamezuje vytvoření jiného výsledného stromu, než jaký je nadefinován v XML schématu. Z toho plyne, že všichni, kdo chtějí daný soubor jakkoliv zpracovat, tak znají tvar stromové struktury, což zabraňuje špatnému zpracování XML souboru.

Jednou z možností použití XML schématu je samozřejmě i použití pro validaci, neboť nám přesně určuje tvar XML dokumentu. Celý tento proces je založen na ověření správnosti kontrolovaného XML dokumentu na základě nějakého schématu (schéma daného XML dokumentu), kde se kontroluje použití atributů, elementů a další omezení definované v XML schématu. Slouží pro ověřování XML dokumentů, které se odněkud stáhne nebo nám někdo pošle nebo si ho uživatel ručně vytvoří, pro následné použití např. v aplikaci nebo na internetu buď jako čtečka nebo úložiště dat.

Některé XML schémata mohou obsahovat u jednotlivých elementů a atributů datový typ. U validace se kontroluje i daný datový typ, při čtení XML dokumentu se může přímo pracovat s daným datovým typem, jako je např. číslo, datum, čas, atp. a ne jen s textovými řetězci. Takový dokumentu se po přidání typů označuje jako PSVI.

Další možností požití XML schématu je využití jako zdroj pro vytvoření XML dokumentu v nadefinovaném tvaru. Celé kouzlo je v tom, že aplikace vygeneruje automaticky XML dokument, jehož tvar bude odpovídat XML schématu. Výsledkem jsou validní XML dokumenty dle zadaného schématu, takže z jednoho schématu budou vždy „stejné“ výstupy, které lze potom použít dále bez validace.

### 2.2.1 DTD

DTD byl prvním jazykem pro popis XML dokumentů, jenž popisoval jazyk SGML. V dnešní době je jeho jedinou velkou výhodou, že ho podporuje velkého množství aplikací.

Jinak tento jazyk má plno nedostatků, nepodporuje určení datového typu pro element a atribut, protože SGML bylo navrženo pro značkování dokumentů, knih, webových stránek, dokumentů, apod., kde se používá, dá se říct, jen text. Jenomže později se XML začalo používat i pro úložiště dat obsahující i jiné datové typy než jen textový řetězec., jako jsou např. čísla, data, url, atd.

Dalším problémem DTD byly jmenné prostory, které dávají šanci použít v jednom dokumentu více sad značek.

Nevýhodou č. 3 je syntaxe jazyka DTD, neboť kód vypadá spíše jako regulární výraz, oproti jiným jazykům definující XML schéma, které jsou ve tvaru XML dokumentu a to má výhodu snadnějšího zpracování.

Jako ukázka je nadefinován Příklad č.3 s elementem zaměstnanec skládající se z elementů titulu, který je nepovinný, jména, příjmení, platu a narození a navíc atributu id, který je povinný.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE zamestnanec [
<!ELEMENT zamestnanec (titul?, jmeno, prijmeni, plat,
narozen)>
<!ATTLIST zamestnanec id CDATA #REQUIRED>
<!ELEMENT titul (#PCDATA)>
<!ELEMENT jmeno (#PCDATA)>
<!ELEMENT prijmeni (#PCDATA)>
<!ELEMENT plat (#PCDATA)>
<!ELEMENT narozen (#PCDATA)>
]>
```

Příklad č.3 Ukázka kódu jazyka DTD

Kromě znaku otazníku, co je u titulu, který značí, že titul může, ale nemusí být použitý, existují i další podobné značky, jednou z nich je třeba hvězdička „*“ představující libovolný počet elementu u kterého je napsána. Dalším znakem je plus „+“ značící, že počet použití daného elementu je minimálně jednou. Pak existuje svislá čára „|“ mezi dvěma elementy, která slouží jako slovíčko „nebo“, což je výběr pouze jednoho z dvojce, oproti čárce „ , “, jenž je jako „a“, takže musí být použity oba elementy kolem čárky, tyto značky lze kombinovat společně se závorkami. [3]

### 2.2.2 XSD (XML schéma)

XML schéma se poprvé objevilo v roce 2001 a bylo přijaé jako uznávaný standard W3C. XML schéma vzniklo z předchozích pokusných jazyků, které se pokoušely nahradit jazyk DTD. Těmito jazyky byly XML-Data a XDR. Dnes je po DTD nejpoužívanějším jazykem definující XML soubory. Využívají ho velké firmy a plno open-source projektů.

XML schéma se označuje zkratkou WXS vzniklou z W3C XML Schema a nebo XSD podle názvu formátu tohoto dokumentu.

XSD odstraňuje všechny nedostatky předchozích XML schémat, tudíž jdou u tohoto schématu použít jmenné prostory, různé základní datové typy, i vlastní datové typy, jež můžou mít i další různá omezení a v poslední řadě je styl kódu napsaný v podobě XML tagu, což umožňuje jednodušší zpracování pomocí parserů.

Jedinou nevýhodou tohoto XML schématu je jeho dosti složitá specifikace a některá poněkud temná zákoutí jazyka., co se téměř vůbec nepoužívají.

#### Datové typy

Datové typy tvoří základ XSD souborů, kde vlastně všechno je datový typ ať už to je základní datový typ, nebo jednoduché či komplexní datové typy. Jednoduchý a komplexní datový typ může být nadefinován jako vlastní datový typ pro pozdější externí použití, třeba pro použití něčeho, co se může ve výsledném XML dokumentu opakovat, jako jsou adresy, osobní údaje, atd. [4]

#### Základní datové typy

Primitivními datovými typy se rozumí předdefinované typy, které mohou být různá čísla podle jejich maximální délky (byte, short, integer, long), jenž mohou být taky bezznaménková nebo s jiným omezením, třeba nezáporné, nekladné atp., další jsou desetinná čísla (decimal, float, double), dále různé modifikace času a data (den, rok, rok-měsíc, čas, časový interval, atp.), logická hodnota TRUE/FALSE, binární data, url, jazyk, obyčejný řetězec a pak mnoho dalších řetězových typů s různýma vlastnostmi. Často se tyto základní datové typy ještě kombinují s dalšími omezeními.

#### Jednoduché datové typy

Jednoduchý datový typ slouží k omezení základních datových typů, na nějž se aplikuje nějaké integritní omezení, které zúží přípustné hodnoty. Definuje se klíčovým slovem `simpleType` , uvnitř je `restriction` s atributem `base` nesoucí nějaký základní datový typ, uvnitř `restriction` jsou různá omezení pro zvolený základní datový typ.

Mezi omezení pro řetězce patří pevná délka (`length`), minimální a maximální délka (`min/maxLength`), regulární výraz (`pattern`), `enumeration`, což je výběr z hodnot, např. muž, žena jako to je v Příklad č.5.

Pro čísla to je minimální a maximální číslice (`min/maxInclusive` a `min/maxExclusive`, kde Inclusive znamená, že číslice ještě patří do oboru hodnot, kdežto u Exclusive daná číslice do oboru hodnot nepatří), počet platných číslic (`totalDigits`) a počet číslic za desetinnou čárkou (`fractionDigits`).

Jednoduchý datový typ lze definovat mimo strom s jménem, jinak řečeno globální deklarace, ten se pak volá u elementu nebo atributu jako typ, tento typ lze volat víckrát, tento typ je ukázaný v Příklad č.4, a nebo uvnitř elementu nebo atributu, jako v Příklad č.5, kde se jméno neuvádí, neboť slouží jen pro daný element nebo atribut.

```
<xs:element name="castka" type="castkaType">

<xs:simpleType name="castkaType">
 <xs:restriction base="xs:decimal">
   <xs:minInclusive value="200"/>
   <xs:maxExclusive value="50000"/>
   <xs:fractionDigits value="2"/>
 </xs:restriction>
</xs:simpleType>
```

Příklad č.4 Jednoduchý datový typ nadefinován jako globální deklarace

```
<xs:element name="pohlavi">
 <xs:simpleType>
   <xs:restriction base="xs:string">
     <xs:enumeration value="muž"/>
     <xs:enumeration value="žena"/>
   </xs:restriction>
 </xs:simpleType>
</xs:element>
```

Příklad č.5 Jednoduchý datový typ uvnitř elementu

**Komplexní datové typy**

Komplexní datové typy slouží k modelování výsledného dokumentu, značí se slovem `complexType`. Vnitřek komplexního typu obsahuje atributy a typ sekvence elementů, mezi které patří `sequence`, `choice` a `all` a někde uvnitř těchto tagu jsou elementy a tyto elementy jsou ve výsledku vlastně potomci elementu, který je předkem daného komplexního typu. Dokument XML obyčejného komplexního typu obsahuje pouze elementy, výjimku tvoří smíšený obsah, jehož výsledkem jsou elementy společně s prostým textem, které mohou být navzájem promíchané, tento styl XML je pouze pro textově zaměřené dokumenty nebo články, nikoli pro datové soubory.

Komplexní datové typy mají jako jednoduché datové typy dvě možnosti definice, a to s jménem mimo hlavní strom, kdy lze tento typ nazvat vlastním komplexním typem, jenž se volá v elementu přes atribut typ, nebo bez jména uvnitř elementu.

### Sekvence elementů - sequence

Sekvence je nejpoužívanějším typem konstrukce uvnitř komplexního typu, označuje se slovem `sequence`. Vlastností sekvence je uložit všechny elementy uvnitř ve výsledném XML dokumentu v takovém pořadí, v jakém jsou uvnitř sekvence nadefinované. Lze nadefinovat i počet použití dané sekvence pomocí `minOccurs` a `maxOccurs`, tudíž kolikrát se tato sekvence má opakovat, implicitní hodnoty obou atributů jsou jedničky, u maximálního použití lze navíc nadefinovat neomezený počet (`unbounded`), což umožňuje mít v XML libovolný počet prvků elementu s touto hodnotou.

```
<xs:element name="golyDomaci" type="golyType"/>
...
<xs:element name="golyHoste" type="golyType"/>

<xs:complexType name="golyType">
 <xs:sequence minOccurs="0" maxOccurs="unbounded">
   <xs:element name="gol">
     <xs:element name="strelecGolu" type="xs:string"/>
     <xs:element name="asistence" type="xs:string"
minOccurs="0" maxOccurs="2"/>
   </xs:element>
 </xs:sequence>
</xs:complexType>
```

Příklad č.6 Ukázka sekvence s počtem zobrazení

V Příklad č.6 je zobrazena sekvence, která může být použita v libovolném počtu, celá sekvence je ve vlastním globálním komplexním typu, což umožňuje tento typ načíst i víckrát, jak je to v této ukázce, kdy je tento typ načtený v elementu `golyDomaci` i `golyHoste`.

### Výběr jednoho z řady prvků - choice

Výběr jednoho z řady prvků se značí tagem `choice` a představuje množinu komponent, nemusí být stejné, v které může být třeba osamostatněný element, nebo klidně i složitější konstukce jako je třeba `sequence` s jakkoliv složitým stromem, ale také to může být klidně i `choice`, atd. Takže tuto množinu tvoří pouze přímí potomci, to co pak obsahují patří do, dá se říct, do jednoho pytle. Hlavním principem `choice` je vybrat pouze jednu komponentu z nadefinované množiny. U `choice` jde taky nadefinovat jako u `sequence` počet použití pomocí `minOccurs` a `maxOccurs`.

```
    <xs:element name="osoba">
     <xs:complexType>
       <xs:sequence>
         <xs:element name="titul" type="xs:string"
minOccurs="0"/>
         <xs:choice>
           <xs:sequence>
             <xs:element name="jméno" type="xs:string"/>
             <xs:element name="příjmení" type="xs:string"/>
           </xs:sequence>
           <xs:sequence>
             <xs:element name="příjmení" type="xs:string"/>
             <xs:element name="jméno" type="xs:string"/>
           </xs:sequence>
         </xs:choice>
         <xs:element name="titul" type="xs:string"
minOccurs="0"/>
       </xs:sequence>
     </xs:complexType>
    </xs:element>
```

Příklad č.7 Ukázka choice konstrukce

V Příklad č.7 je ukázaná konstrukce `choice`, která obsahuje uvnitř dvě sekvence v nichž jsou možnosti na zobrazení jména a příjmení. Ve výsledném XML dokumentu se vybere pouze jedna z těchto dvou sekvencí. Vně jsou nadefinované tituly, které nejsou povinné. Výstupem pak jsou elementy např. ve tvaru titul+jméno+příjmení, titul+příjmení+jméno+titul, jméno+příjmení, příjmení+jméno, a další kombinace.

**Libovolné pořadí elementů – all**

Někdy se může stát, že si chceme pořadí elementů a skupin s elementy určit sami a k tomu nám právě slouží tento typ konstrukce značící se tagem `all`. Tudíž všechny prvky, co jsou uvnitř této konstukce, nemají pevně dané pořadí, ale pořadí si určujeme sami podle našeho uvážení a naší potřeby podle situace. Jeden z problémů u konstrukce `all` je, že u něj nelze definovat libovolný počet použití jako u `choice` a `sequence`, lze pouze nastavit hodnoty 0 nebo 1, ani uvnitř této konstrukce. Dalšími problémy je, že `all` může být použitý pouze uvnitř komplexního typu a nejde kombinovat s `choice` a `sequence`. To znamená, že výsledné XML kód z Příklad č.7, kde je libovolné pořadí jména a příjmení s volitelnými tituly před jménem s použitím `all` jde napsat, tak jak je v Příklad č.8. [5]

```
<xs:element name="uzivatel">
 <xs:compexType>
   <xs:element name="titul" type="xs:string"
minOccurs="0"/>
   <xs:all>
     <xs:element name="jmeno" type="xs:string"/>
     <xs:element name="prijmeni" type="xs:string"/>
   </xs:all>
   <xs:element name="titul" type="xs:string"
minOccurs="0"/>
 </xs:compexType>
</xs:element>
```

Příklad č.8 Příklad na konstrukci all

**Smíšený obsah**

Jak již bylo výše zmíněno, smíšený obsah slouží pro zmixování čistého textu s různými elementy na stejné úrovni v libovolném počtu a pořadí, jinými slovy řečeno, text se může objevit všude mezi elementy, které patří pod daný komplexní typ. Smíšený obsah poznáme, když `complexType` má atribut `mixed` s hodnotou `true`. Lze použít jen v textově orientovaných dokumentech, jako jsou články, různé typy dokumentů, atp., nikoli v datově orientovaných dokumentech.

Dalším tagem po `complexType` musí vždycky následovat `choice`, který obsahuje všechny prvky, skupiny, jenž mohou být v textu zařazeny. U `choice` se většinou nastavuje počet použit `minOccurs` a `maxOccurs`, kde maximum se přiřazuje neomezený počet. Celá tato konstrukce je znázorněna v Příklad č.9, kde ve výsledku mohou být tagy `strong`, `em` a `u` prolnuty textem, tak jako to je v Příklad č.10.

```
<xs:element name="div" maxOccurs="unbounded">
 <xs:complexType mixed="true">
   <xs:choice minOccurs="0" maxOccurs="unbounded">
     <xs:element name="strong" type="xs:string"/>
     <xs:element name="em"/>
     <xs:element name="u"/>
   </xs:choice>
 </xs:complexType>
</xs:element>
```

Příklad č.9 Ukázka smíšeného obsahu

```
    <div>Někde v textu může být <strong>tučný text</strong>.
  Pak může být znovu text a za ním <u>podtžený text</u><em>a za
  ním kurzíva</em></div>
    <div>A nebo může být vložen jen text bez nějakých
  tagu.</div>
```

Příklad č.10 Dvě ukázky výstupu z Příklad č.9

**Elementy**

Prvek `element` tvoří hlavní stavební jednotku celého XML schéma, v XML dokumentu to je značka, jejíž jméno se rovná názvu v atributu `name` v XSD u `element`.

Má dvě možnosti deklarace, tou první je lokální deklarace, což znamená, že element je uvnitř nějakého stromu. Druhou možností je globální deklarace, kdy prvek je potomkem `schema`, na takovýto prvek se lze odkazovat z libovolného místa ve schématu a to pomocí atributu `ref`, kde jako hodnota je jméno globálního elementu, a z toho plyne, že tento prvek nemůže být kořenovým prvkem dokumentu. Ukázka je v Příklad č.11, kde kořenový element je `odstavec` a jeho vnitřní element volá pomocí odkazu element `text`.

```
<xs:element name="text" type="xs:string">

<xs:element name="odstavec">
 <xs:complexType>
   <xs:sequence>
     <xs:element ref="text"/>
   </xs:sequence>
 </xs:complexType>
</xs:element>
```

Příklad č.11 Ukázka elementu s odkazem

`Element` může být beztypový, to je v XML dokumentu element bez textového obsahu, může obsahovat další potomky, výjimkou je potomek `simpleType`, který vlastně definuje základnímu typu nějaké vlastnosti, když potomek je `complexType` a definuje smíšený obsah nebo když element rozšiřuje element `extension`. Typové elementy jsou rozdělené na dvě možnosti. Typ je základním datovým typem, výsledkem je značka s možným textovým obsahem.

Dalším možností je, že v názvu typu je odkaz na vlastní typ, který se vloží dovnitř tohoto elementu jako potomek s celým schématem, co obsahuje. To je zobrazeno v Příklad č.4.

Dalšími možnostmi elementu je počet opakování pomocí `minOccurs` a `maxOccurs`. Dále výchozí hodnota pomocí atributu `default` a nebo zafixovaná výchozí hodnota, která nejde změnit, pomocí atributu `fixed`.

### Atributy

Atribut se značí značkou `attribute`, má stejné vlastnosti jako `element` (`minOccurs`, `maxOccurs`, `default`, `fixed`) a může být deklarován externě i interně.

Předek atributu ve stromě musí být `complexType`, může jim být i `extension`, ale ten patří pod `complexType`, a označuje, že `complexType` nemá žádné elementy. Atribut nemůže obsahovat žádné potomky, kromě `simpleType`, který mu může blíže specifikovat vlastnosti hodnoty atributu. Atribut obsahuje navíc možnost deklarování povinnosti atributu, značí se `use="required"` a znamená, že prvek nesmí zůstat prázdný.

```
<xs:element name="img">
 <xs:complexType>
   <xs:attribute name="src" type="xs:anyURI"/>
 </xs:complexType>
</xs:element>
```

Příklad č.12 Jednoduchá ukázka použití atributu

Výsledkem Příklad č.12 je element img s atributem src, který obsahuje url adresu, výsledk ukázaný v Příklad č.13.

```
<img src="obrazek.jpg"/>
```

Příklad č.13 Výstup Příklad č.12

### Metody návrhů schémat

V předešlých kapitolách byly ukázané různé metody tvorby schémat pomocí lokálních a globálních vlastních typů, elementů a atributů, tyto metody se postupem času stály metody pro správnou tvorbu schémat, a proto dostaly pojmenování.

První metodou je obyčejné sekvenční schéma pojmenované matrjoška, podle tradičních ruských dřevěných dutých do sebe zapadajících panenek a stejný je tak i princip schématu, obsahuje jeden globální element a všechny další elementy jsou uvnitř zanořené, právě jako matrjošky. Ukázka v Příklad č.14, kde je tím jediným kořenovým elementem `clovek` obsahující podelementy `jmeno` a `prijmeni`.

```
<xs:element name="clovek">
 <xs:complexType>
   <xs:sequence>
     <xs:element name="jmeno" type="xs:string"/>
     <xs:element name="prijmeni" type="xs:string"/>
   </xs:sequence>
 </xs:complexType>
</xs:element>
```

Příklad č.14 Schéma typu matrjoška

Dalším používaným typem jsou salámové kolečka, kde jsou všechny elementy nadefinované jako globální, což znamená, že jsou v hlavním elementu pospojovány pomocí odkazů. Výhodou je, že globálně nadefinované elementy jdou použít víckrát. Ale hlavním problémem tohoto principu je, že jeden element nejde použít pro dva rozdílné modely se stejným složením, jako je třeba doktor a pacient, oba mají stejné údaje, to je jméno, příjmení, atd., ale hlavní konstrukce je odlišná, tím jsou myšleny elementy s názvy doktor a pacient. V Příklad č.15 je kořenovým elementem je opět clovek a uvnitř pomocí odkazů specifikuje ostatní elementy, takže jmeno a prijmeni, jenž jsou nadefinované globálně. Ve výsledku vyjde stejné schéma jako je Příklad č.14.

```
<xs:element name="jmeno" type="xs:string"/>
<xs:element name="prijmeni" type="xs:string"/>

<xs:element name="clovek">
 <xs:complexType>
   <xs:sequence>
     <xs:element ref="jmeno"/>
     <xs:element ref="prijmeni"/>
   </xs:sequence>
 </xs:complexType>
</xs:element>
```

Příklad č.15 Schéma typu salámová kolečka

Posledním hojně používaným způsobem je slepý Benátčan, je hodně podobný salámovým kolečkům, ale rozdíl tví v tom, že se nedefinují globálně elementy, ale vlastní typy, tzn. `simpleType` nebo `complexType`, takže elementy jsou definovány lokálně a pomocí jejich typů jsou volány právě ty vlastní typy. Výhodou oproti salámovým kolečkům je, že mohou být globálně nadefinovány dva rozdílné modely v jednom vlastním typu, neboť pojmenování dvou elementů bude odlišné, tak jako v předchozím bodu pacient a doktor, ale v typu budou mít stejný komplexní datový typ pojmenovaný třeba clovek, který je nadefinovaný v Příklad č.16, jehož výsledkem je stejné výsledné schéma, tak jako ve dvou předchozích ukázkách. Jedinou nevýhodou je obtížnost a zdlouhavá definice vlastních typů.

```
<xs:simpleType name="jmenoType">
 <xs:restriction base="xs:string"/>
</xs:simpleType>

<xs:simpleType name="prijmeniType">
 <xs:restriction base="xs:string"/>
</xs:simpleType>

<xs:complexType name="clovekType">
 <xs:sequence>
```

```
    <xs:element name="jmeno" type="jmenoType"/>
    <xs:element name="prijmeni" type="prijmeniType"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="clovek" type="clovekType"/>
```

Příklad č.16 Schéma slepého Benátčana

**Jmenné prostory**

Jednou z dalších výhod oproti DTD je využívaní jmenných prostorů. V XML schéma jsou všechny jmenné prostory deklarovány v kořenovém elementu `schema` jako atributy ve tvaru ve tvaru `xmlns:JmenoProstoru`. Hodně jsou používány i pro externí použití odkazů za pomocí identifikace `target-namespace`. Jmenný prostor se pak používá u elementů a typů pomocí prefixu a to ve tvaru jmenný prostor s dvojtečkou, výjimku tvoří jmenný prostor bez prefixu, to sem pak před elementem nebo typem nepíše nic. V ukázce Příklad č.17 jsou deklarovány dva jmenné prostory xs a tns. Je použita konstrukce slepého Benátčana, u elementů a základních datových typů je použity jmenný prostor `xs` a vlastní komplexní typ `clovekType` se volá v kořenovém elementu s jmenným prostorem `tns`.

```
<schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:tns="urn:x-tvurce:schémata:projekt.1.0">

 <xs:element name="clovek" type="tns:clovekType"/>

 <xs:complexType name="clovekType">
   <xs:sequence>
     <xs:element name="jmeno" type="xs:string"/>
     <xs:element name="prijmeni" type="xs:string"/>
   </xs:sequence>
 </xs:complexType>

</xs:schema>
```

Příklad č.17 Ukázka jmenného prostoru

## 2.2.3 Relax NG

Relax NG vznikl jako reakce na složitost XSD, v roce 2001 byl standardizován sdružením OASIS a od roku 2003 je i normou ISO. Jeho zápis lze psát buď v podobě XML anebo v kompaktní podobě. Relax NG nemá v sobě zaevidované datové typy, ale může je převzít z cizích schémat, ale jdou u něj použít jmenné prostory. Oproti XSD je jednodušší a neskrývá žádné přespříliš utajované věci, tudíž prostý uživatel se ho jednoduše a rychle naučí, proto se čím dál víc stává oblíbenějším, jeho popularita roste a dohání XSD schéma.

Ukázka představuje schéma zaměstnanců, kde každý zaměstnanec má jméno, příjmení, alespoň jeden e-mail, plat, který nemusí mít a id. Speciální znaky hvězdičky, plusu a otazníků zde představují to stejné, co u jazyka DTD. [6]

```
element zamestnanci {
 element zamestnanec {
   attribute id { xsd:int },
   element jmeno { xsd:string },
   element prijmeni { xsd:string },
   element email { xsd:string }+,
   element plat { xsd:decimal }?,
   element narozen { xsd:date }
 }+
}
```

Příklad č.18 Ukázka jazyka Relax NG

### 2.2.4 Schematron

Dalším jazykem je Schematron, který definuje různé podmínky, co musí daný dokument splnit, z toho vyplývá, že takovéto schéma slouží pouze a jen pro validaci dokumentu. Tento jazyk využívá možnosti jazyka XPath.

Žádný z předchozích jazyků neumožňuje například popsat taková omezení, kdy obsah jednoho elementu musí obsahovat větší hodnotu než obsah jiného elementu, nebo vztahy mezi daty uloženými v několika dokumentech.

Schematron je často využíván pro jeho vlastnosti jako doplněk ke klasickým schématům. XSD a Relax NG mají možnost připojit Schematronové schéma přímo do svého schématu. [6]

## 2.3 Java SE

Základy Javy lze nalézt v projektu Oak, který vznikl ve firmě Sun na počátku devadesátých let pro řízení elektronických výrobků. V roce 1994 byl přenesen jako programovací jazyk do prostředí počítačů pod názvem Java (horká káva). Je to vyspělý multiplatformní programovací jazyk, obsahující všechny vlastnosti, které jsou vyžadovány v moderním programování, od modularity programu, řídících konstrukcí, přes silnou typovou kontrolu, multithreading, ošetření výjimek, správu paměti, i silnou podporu pro databáze, XML a síťové operace.

Zdrojový kód je při vývoji přeložen do spustitelného mezikódu (bytecode), který lze pak spouštět pomocí nainstalovaného runtime prostředí JVM (Java Virtual Machine), přímo na různých typech počítačů či technických zařízeních, od Linuxu, Unixu, Windows a další. Většina distribucí Linuxu již v sobě obsahují vývojové prostředky i runtime prostředí Javy, bez nutnosti nového překladu. Na rozdíl od jazyka C, který je rozšířen stejně nebo i více, by přenositelnost

programu měla být dána ne pouze jen na úrovni zdrojového kódu, ale spustitelného programu, jak bylo řečeno výše.

Nižší rychlost, způsobená zpracováním v runtime prostředí může být urychlena s pomocí specializovaných překladačů na cílovém prostředí (Java just-in-time, JIT). I když základní vývojové prostředí obsahuje pouze řádkový překladač, existuje mnoho vývojových nástrojů a rozšíření dalších firem a autorů včetně IDE, i s podporou vývoje GUI aplikací.

Obsah Javy však nelze omezit jen na výčet jejích příkazů. Java je především silně objektová, což umožňuje v ní modelovat, vytvářet, používat a rozšiřovat rozsáhlé knihovny a systémy. Právě objektově je třeba myslet ne jen při psaní programu, ale již při návrhu a analýze. Pro tyto účely byl vytvořen modelovací jazyk UML, slouží k objektovému modelování a popisu konstrukcí reálného světa, převáděných do světa počítačů a informačních systémů. [7]

# 2.4 Swing

Tento pojem představuje v Javě knihovnou pro tvorbu GUI, což je grafický vzhled aplikací. Swing se začal vyvíjet už v roce 1997, kdy společnost Sun přešla z AWT, který je předchůdce Swingu, jehož počátky sahaly do roku 1995. Hlavním důvodem byla závislost AWT na platformě. Swing se stal nedílnou součástí Java SE od verze 1.2. Knihovna Swing je založena na předchůdci AWT

Základní m prvkem Swingu je třída JComponent, z něhož dědí ostatní komponenty, jako jsou tlačítka, textové pole, panel, seznamy, atd. Zatímco různá okna jsou potomky Window.

Velký rozdíl mezi těmito dvěma knihovnami je ve vykreslení komponent, AWT slouží jako rozhraní mezi Javou a grafickou API platformou, takže všechny komponenty vykresluje systém, tyto komponenty jsou závislé na systému. Swing funguje v tomto ohledu jinak, za vykreslení každé komponenty se stará přímo Java a operačnímu systému předá pouze hotový obrázek. Z toho jde krásně vidět, že Swing nezáleží na operačním systému, výjimku tvoří okna, která jsou převzata z AWT, takže okna jsou na systému závislá. [8]

Proto když je Swing nezávislý na operačním systému, tak byla naimplementována možnost na změnu vzhledu celé aplikace pomoci Look and Feel. Tento vzhled se dá změnit i za běhu aplikace. Look and Feel obsahuje vzhledy všech hlavních platforem. Různé vzhledy Look and Feel jdou stáhnout z internetu nebo dokonce si lze nadefinovat vlastní Look and Feel. [7]

# 2.5 SAX

SAX, neboli jednoduché XML rozhraní, je sada abstraktních rozhraní, která umožňují práci s XML dokumentem a to tyto dokumenty číst i vytvářet. SAX vytvořila vývojáři e-mailové skupiny XML-DEV pro programovací jazyk Java. Od tohoto okamžiku se začal rozšiřovat i do ostatních odvětví i mimo tuto platformu. V současné době je k dispozici plno implementací tohoto

rozhraní pro ohromně množství programovacích jazyků a platforem např. specifikace SAX v MSXML 3.0 od společnosti Microsoft, tuto implementaci lze použít v programovacích jazycích C++ nebo VB.

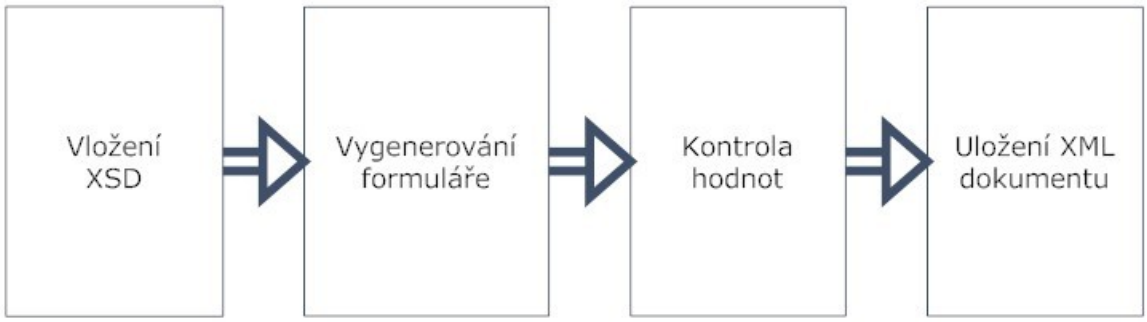
Hlavním principem je způsob proudového čtení, někdy je tento způsob nazývaný událostmi řízené zpracování, jenž postupně čte XML dokument a pro každou část vyvolá událost a naším cílem je napsat obsluhu těchto událostí. Výhodou tohoto způsobu je velká rychlost a malá paměťová náročnost a proto je dost často upřednostňován před standardem DOM, který klade důraz na výkon.

Nevýhodou je, že se dokument zpracovává sekvenčně, což neumožňuje se při čtení vracet, proto u většiny údajů je nutné je průběžně ukládat do paměti nebo ihned zpracovávat. Další nevýhodou je nízkoúrovňové zpracování, tudíž je zapotřebí velké množství pomocných proměnných nebo vlastní nadstavbu. [2]

# 3 Analýza a návrh aplikace

Cílem celé práce bude aplikace, která má generovat formuláře, jejichž struktura odpovídá již výše popsanému XSD dokumentu, který do aplikace vloží uživatel. Pokud vložené schéma je ve správném tvaru, pak uživatel následně vyplní správnými hodnotami tento formulář, který si nechá následně vygenerovat do dokumentu ve formátu XML, ale to jen za předpokladu, že vloží správné hodnoty do formuláře, celý tento běh je zobrazený na Obrázek č.1.

Úkolem této kapitoly je podrobný rozbor, popis a poté návrh celého tohoto projektu, aby byla možná jeho pozdější implementace. Analýza se skládá z několika důležitých částí, kde každá z těchto částí je záměna na nějaký specifický problém, pro nějž se musí najít vhodné řešení, který se musí pro správný návrh vyřešit. První částí je vize, kde se popíše aplikace, její význam, použití, atd. Jednotlivé možnosti uživatele v aplikaci jsou popsány ve funkčních požadavcích s pomocí různých diagramů. Následují analýza řešení s rozebráním vnitřní části aplikace a návrh uživatelského rozhraní.




Obrázek č.1 Jednoduché schéma aplikace

## 3.1 Vize

**Proč?**

Aplikace bude sloužit k vygenerování XML dokumentu z formuláře vytvořeného z XSD dokumentu.

**Co?**

Vloží se data, pomocí vloženého XSD schéma.

**Jak?**

Vygenerují se vložená data v podobě XML dokumentu.

**Kde?**

Práci s aplikací lze provádět na každém uživatelském počítači, jedinou podmínkou je mít nainstalovaný JVM pro zpracování JAR souboru.

**Kdo?**

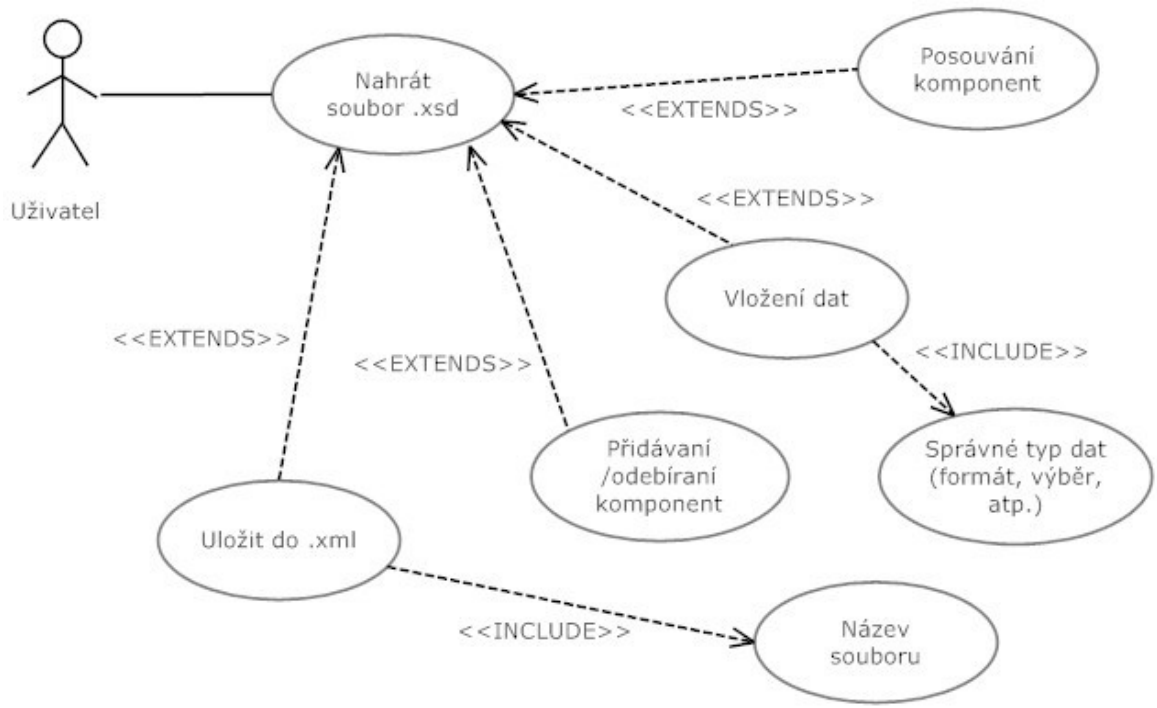
Aplikaci může používat kdokoliv.

# 3.2 Funkční požadavky

Funkční požadavky určují, jaké akce musí být provedeny, co se musí zařídit a různá taková pravidla, pro správný běh aplikace.

## 3.2.1 Use case diagram

Use case diagram, na Obrázek č.2, obsahuje pouze jednoho aktéra a tím je jakýkoliv uživatel používající tuto aplikaci. Uživatel může vložit XSD schéma, na jehož základě se vygeneruje formulář, ale pouze v případě, jestliže je vložený správný dokument. Ve formuláři může přidávat komponenty nebo je posouvat, jen za předpokladu, že jsou tyto možnosti nadefinované ve schématu. Po dokončení vkládaní dat, může tyto údaje uložit do XML dokumentu, ale to pouze, když jsou vložené validní údaje, dle schématu.




Obrázek č.2 Use case diagram

**Nahrát soubor .xsd**

**Typický průběh:** uživatel nahraje do aplikace soubor ve formátu xsd

**Alternativní průběh:** je vložené špatně vytvořené XML schéma, buď není validní anebo obsahuje špatný obsah, tzn. není souborem ve formátu XSD

**Vložit data**

**Typický průběh:** uživatel vloží nové data do vytvořeného formuláře
**Prekondikce:** musí být vložený správný XSD dokument

**Posouvání komponent**

**Typický průběh:** uživatel posune komponenty s touto možností nahoru nebo dolů
**Prekondikce:** musí být vložený správný XSD dokument
**Postkondikce:** záměna dvou sousedících komponent

**Přidávaní/Odebíraní komponent**

**Typický průběh:** uživatel přidá nebo odebere komponentu, tam kde to je možné
**Prekondikce:** musí být vložený správný XSD dokument

**Správný typ dat**

**Typický průběh:** uživatel vloží nebo vybere data ve správném formátu
**Alternativní průběh:** uživatel vloží špatný text, např. místo čísla vloží řetězec

**Uložit do .xml**

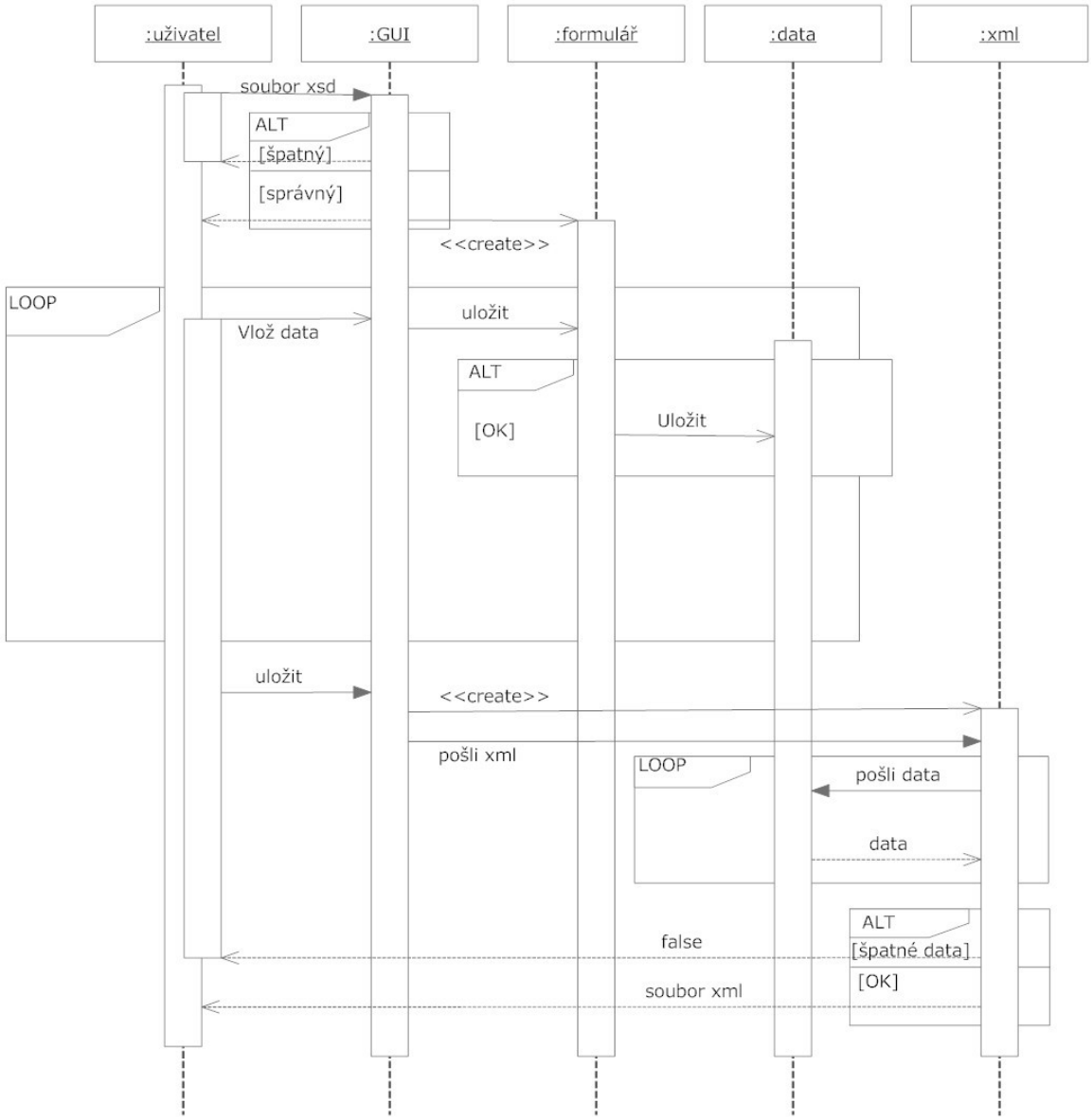
**Typický průběh:** uživatel nechá z formuláře vygenerovat XML soubor
**Alternativní průběh:** chyba při validaci vložených dat
**Prekondikce:** musí být vložený správný XSD dokument
**Postkondikce:** vložení názvu výstupního souboru

**Název souboru**

**Typický průběh:** je vložen název souboru a dokument se následně uloží
**Alternativní průběh:** není vložený název nebo je špatně vložený
**Prekondikce:** uživatel musí vybrat možnost uložení formuláře
**Postkondikce:** uložení výstupního souboru na disk

## 3.3 Sekvenční diagram

Sekvenční diagram zachycuje průběh chování uživatelů a zpracování zpráv aplikace v časové rovině. Zobrazuje posloupnost událostí od začátku do konce za určitý časový interval. Na Obrázek č.3 je znázorněný postup uživatele v aplikaci, kdy musí nejprve vložit správný XSD dokument, jinak není připuštěn do cyklu vkládaní dat do vytvořeného formuláře a až dokončí vkládaní dat, tak si nechá tyto data uložit, program vyšle zprávu pro vytvoření XML dokumentů, jenž si nechá předat data a zkontroluje jejich správný tvar, velikost, formát, atp., při jediné chybě neuloží data do XML dokumentu, ale nechá uživatele pokračovat ve vkládaní dat, pro jejich editaci, tudíž opravu chyb, po opravení všech chyb lze vygenerovat XML dokument a uložit ho na pevný disk počítače.




Obrázek č.3 Sekvenční diagram

# 3.4 Analýza řešení

V předchozích bodech byla popsána aplikace z pohledu uživatele, zde bude popsána podoba uvnitř aplikace, kde jsou popsány různé zádrhely, návrhy, postupy při tvorbě, různé možnosti tvorby a co se musí kde zajistit, aby následná implementace byla v pořádku dle zadání.

Jak již bylo výše zmíněno, že aplikace se bude skládat ze třech hlavních částí, po menší analýze bych to ještě rozšířil o jednu část, tou částí je validace vložených hodnot, ty výše uvedené jsou práce se vstupním souborem, zobrazení daného formuláře a vygenerování výsledného XML dokumentu s vloženými údaji. Tyto částí, tzn. běh aplikace závislé na čase, jsou zobrazeny

v diagramu aktivit na Obrázek č.4. V této sekci je právě prostor, aby se pro každou část udělala důkladná analýza.


Obrázek č.4 Běh celé aplikace na diagramu aktivit.

## 3.4.1 Načtení a práce se souborem

Prvním bodem je vložení správného XSD souboru, tento soubor si musí uživatel vložit ze svého disku z počítače, takže buď to pevný disk, nebo nějaký externí disk. Celé by to mělo být uděláno tak, ať se uživateli zobrazí nějaká komponenta, takže okno, kde bude pouze nabídka XSD souborů a složek pro pohybování se na disku počítače.

### Nalezení jmenných prostorů

Po úspěšném načtení souboru následuje projití a zpracování vloženého schématu, jinak řečeno, začne proces parsování. Ale ještě předtím, než se začne dokument parsovat, tak se musí najít všechny jmenné prostory, protože bez nich, jak bylo výše popsáno v kapitole 0 Jmenné prostory, by nešly nalézt jednotlivé elementy, datové typy, vlastní odkazy, typy, atd., které mají před svým tagem nějaký jmenný prostor, pokud není nadefinován jmenný prostor bez popisku. Z toho všeho plyne, že každé XML schéma může mít libovolný počat jmenných prostorů, takže prvním krokem je tyto jmenné prostory najít a uložit je do nějakého pole, jenž se bude později procházet a hledat tagy s těmito jmennými prostory, takže výsledkem bude něco ve tvaru jmenný prostor dvojtečka a název, třeba `xs:element` nebo `xs:string`. Všechny jmenné prostory najdeme v kořenovém elementu `schema`, kde jsou nadefinovány ve tvaru `xmlns:JmenoProstoru`.

### Kořenový element

Dalším problémem je nalezení kořenového elementu pro výsledný XML dokument. Ten najdeme na 2. úrovni ve stromě, hned po kořenovém elementu `schema`. Kořenovým prvkem může být pouze `element`, tudíž ostatní prvky jsou pouze vlastními typy nebo odkazy. A tak se bude hledat nějaký tvar elementu, jež splňuje různá pravidla, aby mohl být kořenovým prvkem. V kapitole 0 Metody návrhů schémat jsou popsány různé typy návrhů schémat, z nichž zde bude odvezený kořenový element.

A protože má být kořenový prvkem element, tak začneme u salámových koleček, kde globální typy jsou právě elementy. Takže cílem je tady najít právě ten jeden jediný kořenový element a ten se pozná tak, že jeho jméno nefiguruje v žádném jiném elementu jako odkaz ref, tudíž si na začátku parsování uložíme všechny odkazy ref do nějaké jednoduché kolekce, pomocí niž potom zjistíme, zda je element použitý někde jinde, to znamená, že není kořenovým elementem, proto se elementu uloží i s podstromem do nějaké jiné kolekce jako odkaz pro pozdější použití. Pokud název elementu není v poli rref odkazů, tak je právě tento element našim kořenovým.

Další návrhovou metodou byla metoda slepého Benátčana, kdy elementy pomocí typů volají vlastní typy, z toho plyne, že kořenový element obsahuje v typu odkaz na existující vlastní typ, a proto je tento element kořenovým elementem. Pro názvy vlastních typů by mělo být vytvořeno taky samostatné pole na jejich kontrolu, zda typy použité v elementech existují.

Poslední konstrukcí, která je výše popsána, jsou matrjošky, jejich deklarace tkví právě v jednom kořenovém elementu a žádném jiném globálním typu či odkazu. Ttento element obsahuje pouze název a uvnitř má lokální jednoduché a komplexní typy obsahující atributy a elementy.

Do speciální kategorie bych zařadil samotný element s jménem a s obyčejným základním typem a element obsahující čistě jen lokální jednoduchý datový typ, jejich výsledný XML dokument by měl vypadat jako v Příklad č.19.

```
<nazev>Text</nazev>
```

Příklad č.19 Výstup kořenového elementu s textem

Na závěr k nalezení kořenového elementu si řekneme shrnutí k této problematice, jenž nám říká, že každé schéma nemusí být napsáno jedním z ukázaných typů návrhu, ale tyto metody mohou být kombinovány, a tak může být hlavním kořenovým elementem kořen z matrjošky, jež bude obsahovat odkaz na salámové kolečko nebo slepého Benátčana, ba dokonce na oba dva, či na jednoho z těch dvou, který bude mít ve své deklaraci toho druhého, což ukazuje, jak může být XML schéma libovolně napsané, a jak není jednoduché ho složit dohromady.

**Vytvoření stromu**

Při procházení vloženého XSD souboru, se jednotlivá data uloží do stromů, takže třeba nějaký list, nemusí být ani seřazený. Při tomto ukládaní nesmíme zapomenout na to, že je jeden kořenový strom, který může obsahovat odkazy na globální vlastní typy, což by vlastně měly být další stromy, a proto tyto vlastní typy, by měly být uloženy v nějaké mapě a opět nemusí být seřazená, kde klíč by měl být název vlastního typu a jako hodnota bude třída se stromovou strukturou odpovídající danému vlastnímu typu.

Výsledná stromová struktura se tedy složí v jeden výsledný strom a to z kořenového stromu a globálních vlastních typů.

## 3.4.2 Vygenerování formuláře

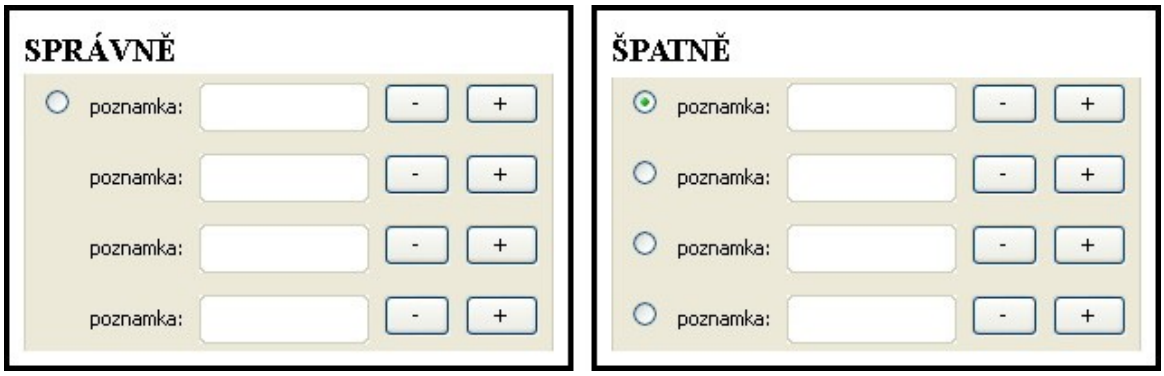
Dalším bodem je vygenerování formuláře, vybraným tagům se přidělí jim odpovídající komponenty, tyto komponenty budou rovněž uloženy ve stromové struktuře pro pozdější potřebu tyto data získat, díky odkazům na tyto komponenty uložených právě ve stromové struktuře.

**Řešení**

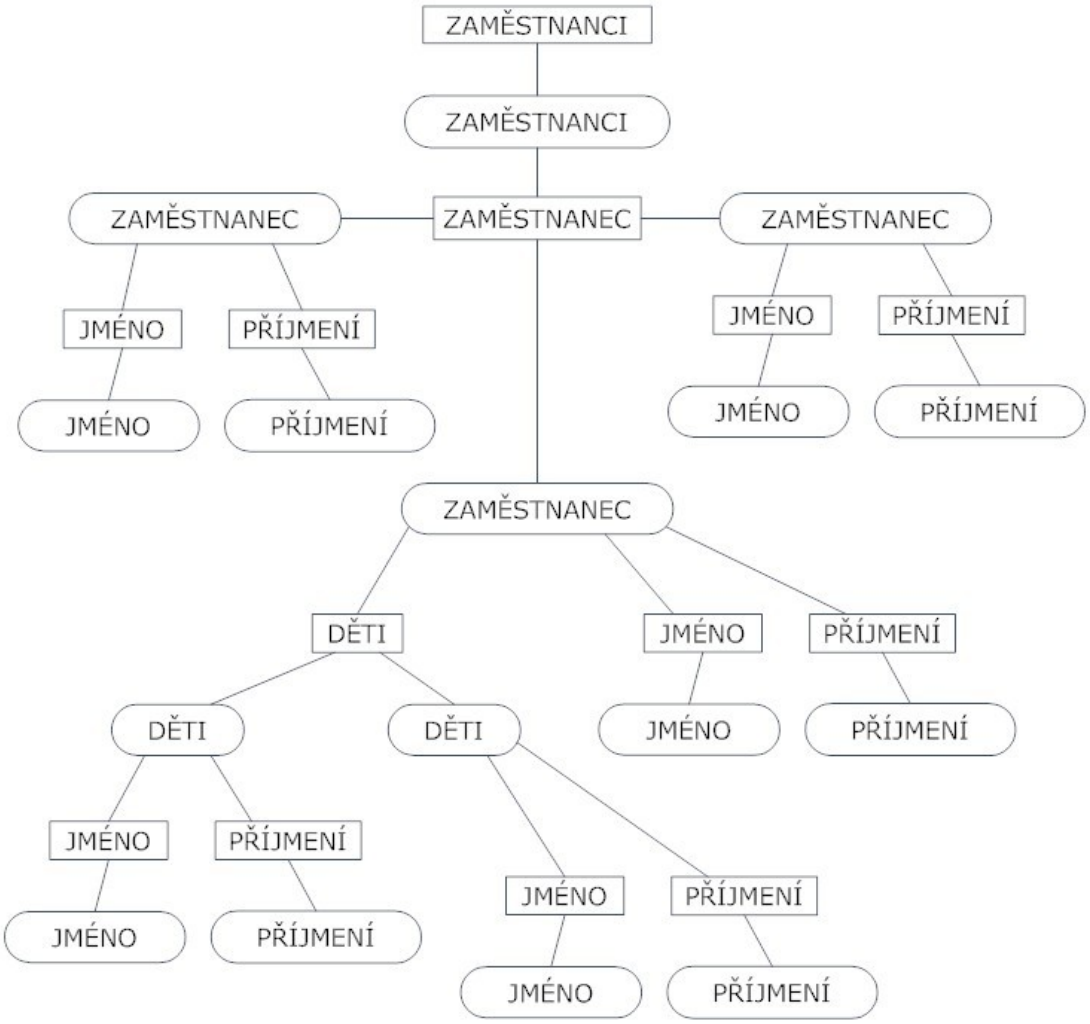
Ono když se zamyslíme, tak zjistíme, že strom vytvořený ze schématu zobrazuje pouze sloučené vlastní typy s kořenovým elementem, tudíž složenou strukturu dané vloženého schéma, ale pro vygenerování formuláře, tento stromu bude nutné pozměnit a to kvůli nastavenému počtu elementu, tím je myšleno množství komponent odpovídající v XML schéma atributu minOccurs a maxOccurs, proto je nutné strukturu pozměnit. Například když je element vytvořen 4x, tak musí každý z nich mít podle původního stromu potomky stejného typu a to právě znamená, že to nesmí být tytéž elementy, nýbrž odlišné, tudíž nové objekty.

Bude k tomu zapotřebí dvou tříd, i když to vypadá že může být pouze jedna třída s listem potomků téže třídy, tak zdání klame, protože třeba u tagu choice, kde se vybírá pouze jeden z potomků, to neplatí, protože potomek element nemusí mít vždycky maxOccurs roven jedné, ale

bude zobrazen třeba 4x, takže kdyby to bylo pouze v jedné třídě, tak by tyto čtyři elementy nebyly jako jeden celek, ale další komponenta v choice na výběr, což je špatně, proto se budou potřebovat dvě třídy, které budou propojeny. Toto je názorně ukázáno na Obrázek č.5, kde element poznámka s neomezeným počtem použití, vlevo je správný výstup, kde celá skupina má jedno přepínací tlačítko, vpravo má každá komponenta přepínací tlačítko, tudíž v tomto případě lze vybrat pouze jednu z těchto poznámek, nikoliv celá jejich skupina.




Obrázek č.5 Správné a špatné vygenerování choice s neomezeným počtem




Obrázek č.6 Ukázka stromu tříd se zaměstnanci

Na Obrázek č.6 je nastíněn celý zmíněný problém, kde je vždycky ke každému tagu pro daného předka vytvořen vytvořený nový objekt, který v nějakém listu nese daný počet vytvořených prvků daného tagu jako objekt s vytvořenou komponentou, potomci těchto komponent jsou právě další tagy. Na tomto obrázku to jde hezky vidět, kde každá část tagu se skládá ze dvou částí, první jsou hranaté obdélníčky, které nesou odkazy všechny své prvky v objektech s komponentami, což jsou kulaté obdélníčky, které obsahují komponentu a list s potomky.

### Typy komponent

Cílem práce je vytvořit formulář pro uživatele, do kterého si poté budou vkládat své informace, údaje, hodnoty, a tak aplikace musí obsahovat komponenty pro tyto informace. To neznamená, že budou všude vytvořené jen obyčejné textové pole sekvenčně za sebou, právě naopak, existuje několik prvků, které budou v aplikaci využity.

V jednoduchém datovém typu je možnost použití výčtu hodnot pomocí enumeration, určitě by nebylo v pořádku, kdyby se uživateli zobrazilo obyčejné textové pole a očekávalo se, že vloží jednu z nabízených hodnot, mnohem lepší je použití komponenty, která dané hodnoty nabízí, skvělou volbou bude otvírací seznam s danýma hodnotami. To samé platí pro datový typ boolean, kde je možnost true/false, proto se pro boolean použije zaškrtávací políčko. Pro textový řetězec string, jenž může obsahovat i řádky, se použije TextArea. Pro typy dat a časů bude vytvořená speciální komponenta nabízející daný typ. Pro zbytek datových typů bude použité obyčejné textové pole.

V konstrukci all je možnost seřazení prvků do libovolného pořadí, to je nutné nějak zařídit, nejlepší možností bude použití tlačítek nahoru a dolů, pro každou komponentu uvnitř zmíněné konstrukce. Výjimkou je první komponenta, která nejde posunout výše, proto nebude vlastnit tlačítko nahoru, to stejné bude platit pro poslední prvek, jenž bude bez tlačítka dolů. Tlačítko se budou využívat i pro elementy s počtem komponent daného elementu a to tak, když je maximální povolený počet prvků větší než je, tak se zobrazí tlačítko pro přidání dalšího prvku, a když je počet prvků větší než minimální počet, tak zase tlačítko pro odebrání prvků.

Další konstrukcí je choice, jeho cílem je výběr jednoho z potomků, tudíž u každého jeho potomků bude přepínací tlačítko pro jejich výběr.

Element bez základního typu, který obsahuje jiné podelementy musí být taktéž nějak označen, aby šlo vidět, kam potomci patří. Nabízí se možnost hodit všechny potomky do nějakého rámečku, úplně nejvhodnější volbou je rámeček s titulkem.

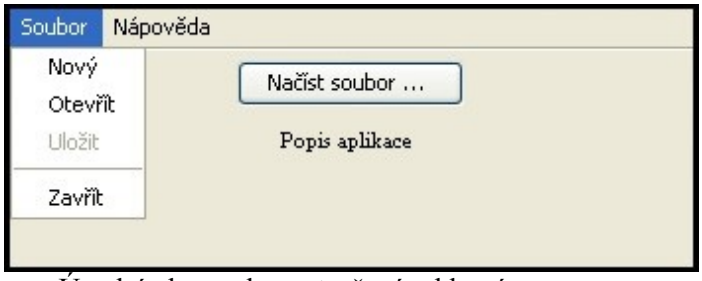
### 3.4.3 Kontrola hodnot a uložení do XML

Před uložením informací do XML dokumentu, se musí nejprve provést validace hodnot. K tomu se použije strom s komponentami, kde všechny atributy a elementy s typem nesou hodnotu. Prvním krokem validace je kontrola datového typu, poté následuje kontrola číselných hodnot, to jsou kladné, nezáporné, záporné a nekladné hodnoty, následují všechny bezznaménkové číselné typy. Dalším krokem se bude kontrolovat, zda byl datový typ nadefinován pomocí jednoduchého datového typu a pokud ano, tak jestli byly u něho nastavené další omezení pomocí restrikce, to je třeba délka, minimální číselná hodnota, počet číslic a další, viz kapitola 0 Jednoduché datové typy. Po každé zkontrolované hodnotě se element nebo atribut s hodnotou uloží do stromu XML, při chybě se XML dokument neuloží a zobrazí se červeně komponenta ve formuláři, v niž je chybná hodnota. Pokud celá validace proběhne v pořádku, tak bude uživateli nabídnuta komponenta jako pro načtení souboru, ale s tím rozdílem, že se soubor nebude načítat, ale uloží dokument na pevný disk, v komponentě půjdou vidět jen XML dokumenty.

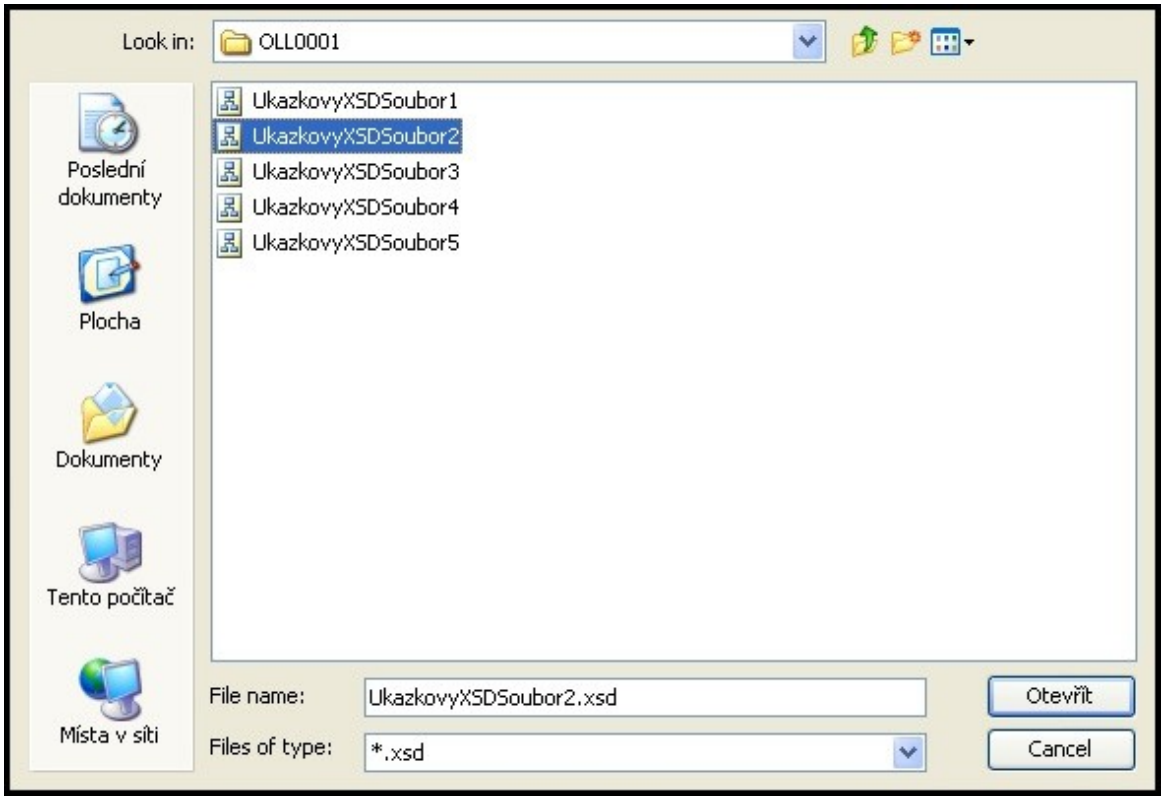
## 3.5 Uživatelské rozhraní

Uživatelské rozhraní bude tvořit hlavní okno, jenž se bude skládat ze dvou základních částí, hlavní menu aplikace a střed aplikace pro zobrazení výstupu.

V menu programu budou tlačítka s různými akcemi, patří tam otevření úvodní obrazovky, na které bude krátký popisek s tlačítkem pro otevření nového XSD soubore, toto tlačítko bude umístěno staticky i v hlavním menu aplikace, jako reakce na to se otevře nové okno s průzkumníkem složek a souborů XSD zobrazeného na Obrázek č.8, kde bude možnost výběru souboru XSD pro otevření a vygenerování formuláře.

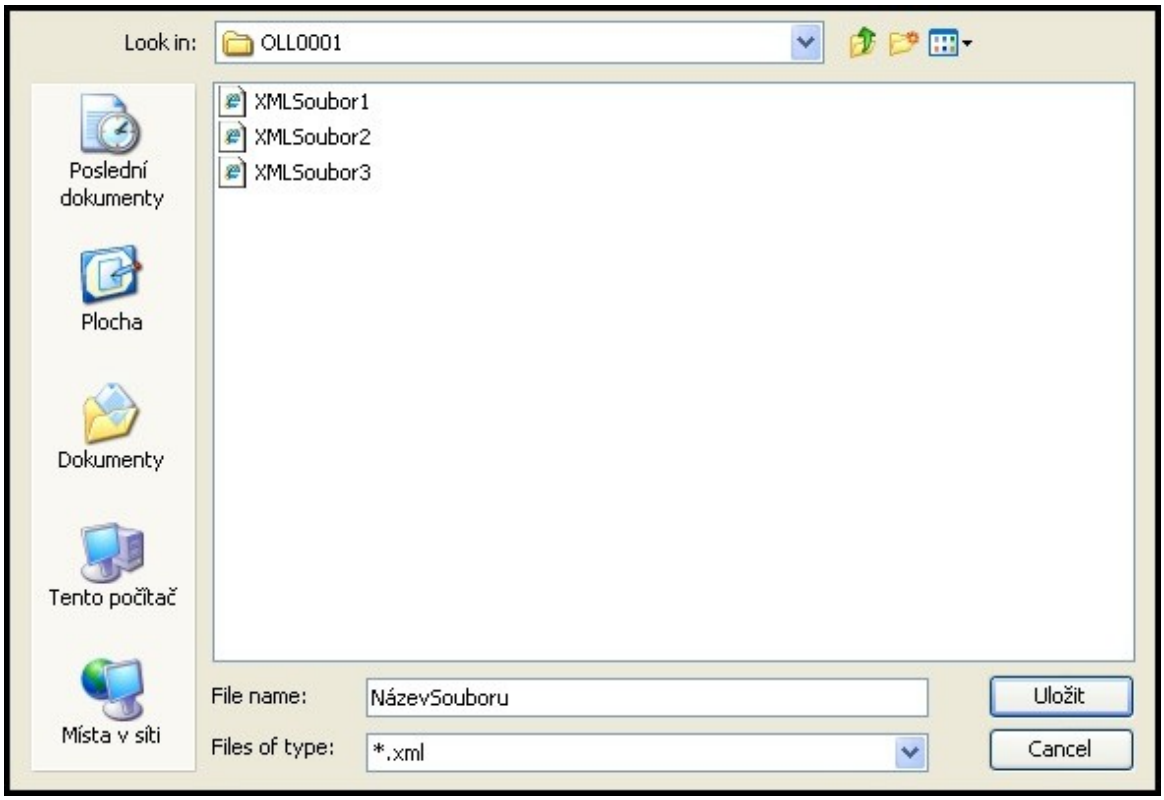



Obrázek č.7 Úvodní obrazovka s otevřeným hlavním menu




Obrázek č.8 Průzkumník pro načtení souboru.

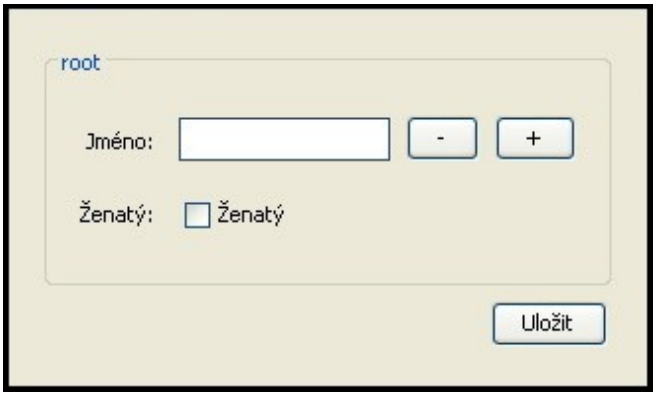
Dále bude v hlavním menu tlačítko na uložení, toto tlačítka bude aktivní, jen v případě načteného souboru XSD, neboť se nemůže nic uložit, když není nic otevřené, co by se zpracovalo. Úkolem tlačítka uložení je otevření nového okna s průzkumníkem, kde budou opět složky a soubory XML jako na Obrázek č.9, jenomže tady se očekává vložení názvu pro vygenerovaný XML soubor, který se do dané složky uloží pod příslušným názvem.




Obrázek č.9 Průzkumník pro uložení výsledného XML dokumentu

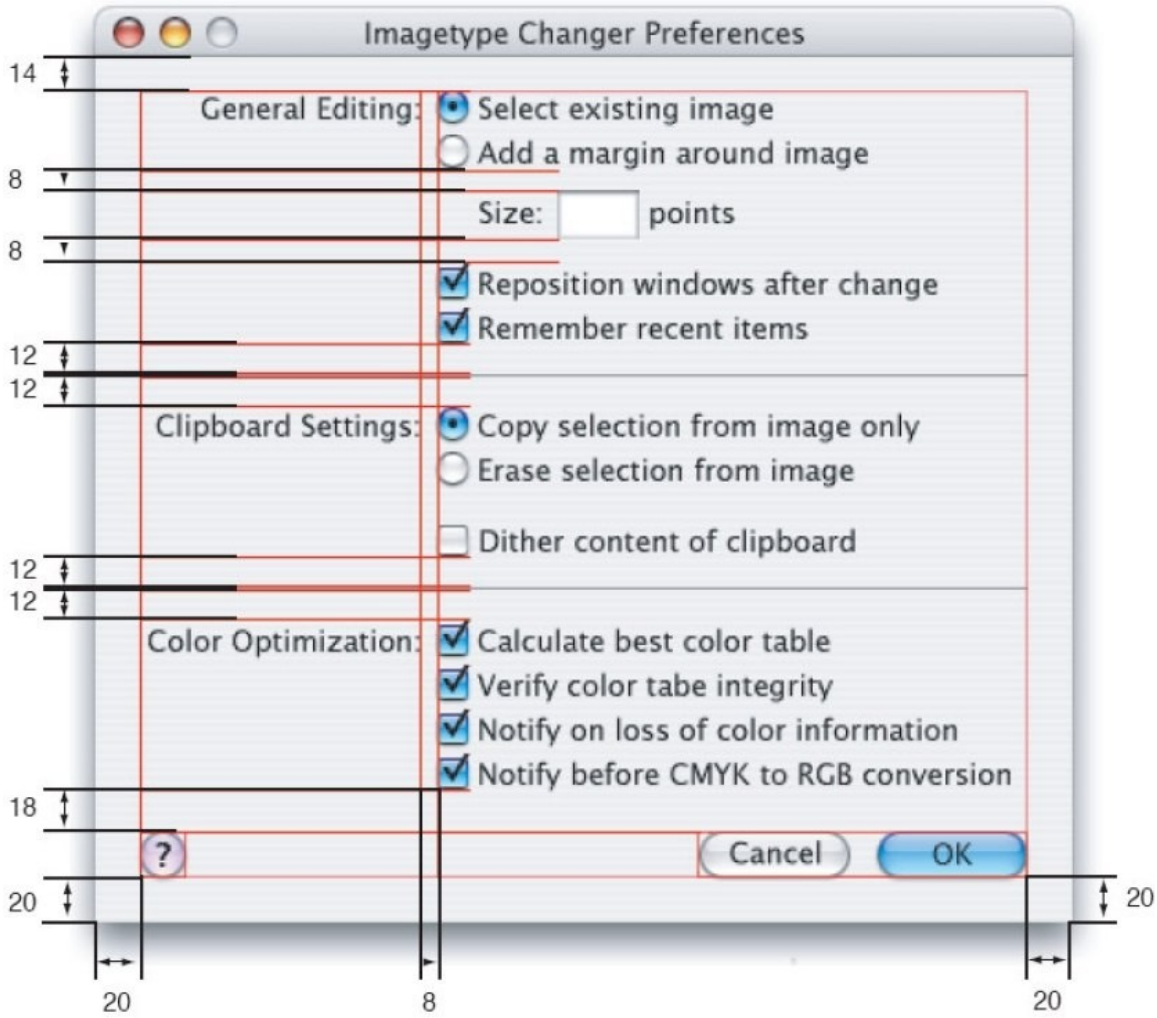
Posledními tlačítky v hlavním menu bude tlačítko zavřít, což ukončí a zavře celou aplikaci. A ještě v novém panelu tlačítko s odkazem na nové okno s informacemi o programu, autorovi, atp.

V hlavní části aplikace, to je uprostřed, což je pracovně proměnná oblast aplikace, bude na úvodní obrazovce, jak bylo již výše zmíněno, tlačítko pro otevření XSD souboru a nějaký krátký popisek. Po otevření vybraného XSD souboru se na tomto místě vygeneruje formulář dle XSD souboru. Podle datových typů se můžou vygenerovat komponenty popisku, textového pole, přepínací políčka, otvírací seznamy, zaškrtávací políčka, atd.




Obrázek č.10 Hlavní oblast aplikace s vygenerovaným formulářem a tlačítkem uložit

Cílem není pouze tyto všechny komponenty naházet do nějakého okna, ale taky je nějak šikovně umístit, ať má aplikace a hlavně zobrazený formulář nějaký systém. Takže všechny vygenerované komponenty a nejen ty, se budou řídit tzv. firemními pravidly, kde jsou popsány vzdálenosti různých komponent od sebe, vzdálenost od okraje, a plno dalších. Všechno je zobrazeno na Obrázek č.11.




Obrázek č.11 Rozložení komponent, dle firemních pravidel [9]

# 4 Implementace

V kapitole implementace jsou popsány nejdůležitější třídy a metody celé aplikace a jejich komunikaci mezi nimi a výše rozebrané problémy převedené do programovacího jazyka. Pro tvorbu aplikace je použitý programovací jazyk Java SE a vývojové prostředí NetBeans [10] s verzí 7.0.1.

## 4.1 Hlavní okno

Hlavní třída se spouštěcí metodou main najdeme ve třídě App.java, což je vygenerovaná třída v NetBeans, kde na tuto třídu navazuje třída View.java. View dědí základní komponentu celé aplikace a tou je hlavní okno FrameView, které obsahuje všechny statické komponenty a ovládá všechny změny zobrazování hlavní části aplikace pomocí metody setComponent(Component c), která je součástí zděděné třídy FrameView, kde jako parametr předávám vždycky panel s ostatními komponentami.

## 4.2 Načtení a uložení souboru

Načtení a uložení souboru je provedeno stejným způsobem, otevře se okno, v němž jsou jednotlivé soubory a složky na disku, to zajišťuje objekt třídy JFileChooser. Pomocí souborového filtru, jenž obsahuje třídu dědící z FileFilter, díky ní pak vidíme jen vybrané soubory, u otevření to jsou sobory s koncovkou XSD, při uložení zase soubory XML. Filtr projde všechny soubory v otevřené složce a každého z nich prožene metodou accept, která vrací buď true nebo false. A to díky podmínce uvnitř, která kontroluje, zda je koncovka souboru rovná XSD nebo XML, což je hodnota true, která nechá soubor zobrazit. Zobrazit se musí i všechny složky, kvůli pohybu na disku, to zařizuje metoda isDirectory(), která je součástí souborové třídy File.

## 4.3 Výběr dat ze souboru XSD

Všechny údaje a informace ze souboru XSD jsou získávané pomocí SAX parsru, a to díky jeho rychlosti, menšími nároky na paměť a pro mě přehledně psaným kódem. Všechno toto je použito v třídě MujHandler.java ležící v balíčku cteniSouboru společně s dalšími ostatními pomocnými třídami.

### 4.3.1 Začátek parsování

V analýze jsou rozebrány problémy ohledně jmenných prostorů a hlavně problém vlastních typů a odkazů, to znamená, že musí být v průběhu parsování předem známé jejich názvy, a proto na začátku dokumentu se zavolá jiný parser s názvem MujHandlerNajdiRootAVlastniTypy, aby

tyto všechny názvy našel a uložil do listů a to zvlášť odkazy, vlastní komplexní typy i jmenné prostory.

### 4.3.2 Procházení dokumentu

Samotný dokument se prochází pomocí metod startElement a endElement. V obou metodách je cyklus na procházení všech jmenných prostorů a plno podmínek na nalezení všech tagu.

V této části programu jsou použité dvě možnosti pro uložení elementů, jelikož existují dva typy, a to hlavní strom s kořenovým elementem, který je jen jeden, takže stačí obyčejný neseřazený list, ArrayList, pro vlastní typy a odkazy tu je hash tabulka, kde klíč je název vlastního typu a hodnotou je odkaz na objekt, jenž v sobě drží strom pro daný typ či odkaz.

**Uložení kořenového elementu, vlastních typů a odkazů**

Prvním problémem jsou kořenové elementy dokumentu a vlastních typů. Kvůli tomu je vytvořená pomocná proměnná jménem element, je ukázána v Příklad č.20, do niž se na začátku všech tagů vloží jejich informace v podobě třídy Elementy a na konci se tyto informace odebírají a vloží se místo nich předek, kořenový element má jako předka null hodnotu, podle ní se pozná kdy je další tag kořenovým tagem. Tato pomocná proměnná slouží pro orientaci ve stromě.

Když je tato proměna prázdna, tak se hledají kořenové elementy. Pokud je to jiný tag než element, tak je to vlastní typ. U elementu se první projde, zda není jeho název v listu s odkazy ref, pokud je, tak je to to vlastní typ. Když není odkazem, tak se bude dále kontrolovat, zda jeho typ leží v listu vlastniCT, což jsou vlastní komplexní typy, pokud ano, tak je hlavním kořenovým elementem dokumentu, to znamená, že se vloží do listu pro výsledný hlavní strom pojmenovaný data. Když nemá žádný typ a neslouží jako odkaz, tak je taktéž hlavním kořenovým elementem dokumentu, což představuje návrhový typ matrjoška. Deklarace všech těchto listů je v Příklad č.20.

```
private ArrayList<Elementy> data=
              new ArrayList<Elementy>();//hlavni strom
protected Hashtable<String,VlastniTyp> vlastniTypy=
              new Hashtable<String,VlastniTyp>();
private List<String> refy=new ArrayList<String>();
private List<String> vlastniCT=new ArrayList<String>();
private List<String> ns=new ArrayList<String>();
private boolean vlastniTypBool=false;
private Elementy element;//pomocna promenna pro orientaci
```

Příklad č.20 Deklarace důležitých listů, hash tabulky a proměnných

Pokud je zjištěný nějaký vlastní typ nebo element či atribut jako odkaz, tak pomocnou proměnou simpleTypeBool představující vlastní typ nastavím na true, pomocí niž v programu rozeznám strom pro vlastní typ. Vlastní typy a odkazy vytvoří odkaz na vlastní typ, kde je list pro

strom celého vlastního typu, který je následní uložen do hash tabulky, jenž obsahuje jako klíčovou hodnotu název vlastního typu či odkazu, opět je tato hash tabulka i pomocná proměna zobrazena v Příklad č.20.

**Tagy uvnitř stromů**

Když je nalezen vlastní typ nebo kořenový element, tak následuje kontrola dalších tagů, hlídá se každý tag, pro nějž je vytvořená třída na výběr potřebných údajů z tagu, kde se následně vytvoří objekt třídy Elementy, která je společná pro všechny, s touto třídou se pracuje po celou dobu běhu aplikace, obsahuje všechny informace o tagu včetně jeho předka i jeho potomků. Po vytvoření instance Elementy se zjistí, z výše zmíněné proměnné představující vlastní typ, hodnota, podle niž se tato instance uloží na konec buď do hlavního stromu, anebo do posledního vytvořeného objektu vlastního typu do jeho vnitřního listu na konec tohoto stromu pomocí metody pridat. Metoda pridat i list jsou zobrazeny v Příklad č.21.

```
public class VlastniTyp
{
    protected ArrayList<Elementy> data=
                new ArrayList<Elementy>();
    ...
    public void pridat(Elementy prvek)
    {
        data.add(prvek);
    }
    ...
}
```

Příklad č.21 Třída VlastniTyp s metodou pro přidání elementu do listu

# 4.4 Sloučení stromů

Na konci celého parsování vznikne hlavní strom, který může obsahovat odkazy na vlastní typy, které se musí sloučit do jednoho stromu. To je provedeno ve třídě slouceni.java z balíčku cteniSouboru, kde tuto akci provádí metoda sloucit, její atributy je list s hlavním stromem, hash tabulka s vlastními typy a list s jmennými prostory, jako návratová hodnota je list se sloučeným výsledným schématem.

Hlavním jádrem je procházení listu s hlavním stromem, kde se kontrolují u elementu a atributů, zda obsahují odkaz ref, nebo jestli mají v typu vlastní typ místo základního. Když se jedná o jednoduchý datový typ, tak se jeho obsah překopíruje do předka, tudíž elementu nebo atributu. Pokud se nejedná ani o jedno, tak se pouze jejich odkaz překopíruje do výsledného listu. V případě, že se jedná o odkaz nebo vlastní typ, tak se zavolají metody pro následné sloučení.

### 4.4.1 Metody pro sloučení vlastních typů a odkazů

Pro každou z těchto dvou akcí se používá jiná metoda, ale obě si jsou hodně podobné. Jako parametry se jim posílají výsledný list, předek, jmenné prostory a hash tabulka s vlastními typy. Rozdíl těchto dvou metod je v tom, že při přidávaní vlastního typu, se do elementu či atributu pouze připojí, u odkazu ref se předek sloučí s prvním tagem.

**Sloučení vlastního typu**

První se najde potřebný strom vlastního typu v hash tabulce, poté se projdou všechny tagy nalezeného stromu. U prvního tagu stromu vlastního typu se nastaví jako předek parametr předka, což je vlastně to naše sloučení. U každého tagu se hledá, zda neobsahuje odkaz ref a nebo se nemá sloučit s jiným vlastním typem, to by znamenalo rekurzi, jinak se tento tag připojí do výsledného stromu.

**Sloučení odkazu ref**

Jediný rozdíl je v tom, že se nepředává odkaz na předka prvnímu tagu, ale druhému, neboť obsah první tagu se právě překopíruje do předka, většinou jde jen o základní datový typ, ale ne vždy tomu tak je. Proto je první tag uložen do proměnné a celý je překopírován do předka. U všech dalších potomku tohoto stromu z vlastních typů, se zkontroluje zda je jeho předek roven proměnné prvního předka daného stromu, pokud jsou si rovny, tak to znamená, že daný tag je na druhé úrovni, tudíž se mu zamění předek za předka z parametru, celý tento postup je ukázaný v Příklad č.22, důležité části kódu jsou tučným písmem, z kódu jsou vybrány jen ta část odpovídající práci s předky.

```
private void projdiRef(List<Elementy> l, Elementy el,
List<String> ns, Hashtable<String, VlastniTyp>
vlastniTypy) throws Exception
{
 Iterator<Elementy> it=vt.getData().iterator();
 boolean prvni=true;
 Elementy predek=el;
 while(it.hasNext())
 {
   if(prvni)
   {
     predek=e;
     první=false;
     //předání hodnot a kontrola refu a typů
   }
   else
   {
     if(e.getRodic().equals(predek))
```

```
                e.setRodic(el);
              //kontrola refu a typů
            }
          }
        }
```

Příklad č.22 Sloučení refu, kontrola předka

# 4.5 Vytvoření formuláře

Pro vytvoření formuláře byla vytvořena třída VytvorFormular.java v balíčku vytvoreniFormulare, na jeho začátku než se začnou tvořit instance všech komponent, je nutné, aby byl vyřešený problém ohledně rozložení hlavního stromu.

## 4.5.1 Vytvoření stromu pro formulář

Problém s vytvořením formuláře z jednoduchého stromu se strukturou složeného XSD dokumentu byl popsán v kapitole 0 Řešení, tudíž místo stromu s jednou pomocnou třídou, byl vytvořený strom skládající se ze dvou tříd, schéma těchto tříd bylo nastíněno na Obrázek č.6.

Celý tento problém je vyřešený v metodě projdi a vytvorSchema uvnitř objektu VytvorFormular. Kde se postupně projdou tagy z původního stromu, kde každý tag toho předcházejícího je nahrazen objektem třídy PrvkySchema.java, na Obrázek č.6 tento objekt představuje hranaté rámečky, jenž obsahuje předka a list s instancemi třídy PrvkyElementu, který představuje množství daného tagu, v XSD dokumentu to je zaznačeno minOccurs a maxOccurs, na Obrázek č.6 to jsou zakulacené rámečky, kde jde například vidět, jak hranatý zaměstnanec obsahuje 3 objekty zaměstnanců a všichni mají stejné potomky, jméno, příjmení a libovolný počet dětí, ale každý z potomků musí být jiný objekt, ať můžou mít i jiné data. Ve výsledku to znamená, že každý PrvekElementu je vlastně jedna komponenta ve formuláři, třeba každé jméno a příjmení bude textové okno a zaměstnance, zaměstnanci, dítě budou zase panely s rámečkem s popiskem. Z obrázku jde ještě vidět, jak každý objekt s komponentami obsahuje potomky, které jsou uloženy v neseřazeném listu, tudíž je použitý ArrayList.

Celý tento cyklus probíhá v metodě projdi, její deklarace je v Příklad č.23, jako parametr je předek, tudíž objekt PrvekElementu s vytvořenou komponentou. V této metodě se vytvoří hlavní PrvekSchema a cyklem podle počtu tagu se vytvoří jeho všechny prvky pomocí PrvkyElementu, kde na konci každého cyklu se volá rekurze metody projdi, tímto způsobem je vytvořený strom složený z objektů dvou tříd.

```
public void projdi(PrvkyElementu pe2){...}
```

Příklad č.23 Deklarace metody projdi

### 4.5.2 Vytváření komponent

Při vytváření stromu, popsaného v předchozí kapitole, se v cyklu vytváření daného počtu instancí PvrkuElementu zavolá i metoda pro vytvoření komponenty odpovídající danému tagu. V každém cyklu je volána metoda vygenerujKomponentu s parametrem Elementy nesoucí informace daného tagu. Uvnitř této metody se pomocí podmínek hledají základní datové typy a jejich modifikace, tak jak jsou popsány v kapitole 0 Typy komponent. Všechny vytvořené komponenty mají stejného předka, a tím je JComponent, tudíž všechny komponenty jsou uloženy v proměnné s typem JComponent. Výjimku tvoří komponenty kombinace dat s časem a komponenty choice a potomky choice.

**Datum a čas**

Pro data s časem, jenž jsou jejich typy a kombinace obsaženy v osmi datových typech, jsou vytvořeny 2 třídy, obě dědí z JPanelu, v první třídě DateTime se zjistí jaký to je typ a jaký kód se má odeslat do druhé třídy DateTimeComponent, ve které je do JPanelu přidán Spinner a v určitých kombinacích i časové pásmo, jehož představuje otvírací seznam. Spinner a otvírací seznam je vytvořen až po připojení JPanelu k potomkovi při vykreslování formuláře, neboť kdyby byly vytvořený dříve, tak by se pak překreslily na prázdný JPanel, a to je uděláno tak, že DateTime připojí k sobě předem vytvořenou instanci DateTimeComponent.

Podle definice musí mít každý datový typ s datem nebo časem přesný formát podle definice XSD dokumentů. Všechny tyto formáty musí být dodrženy, takže například datový typ gMonth musí být ve tvaru --05, což znamená květen a musí obsahovat i ty dvě pomlčky. V Tabulka č.1jsou znázorněny všechny typy dat a časů.

| Datový typ | Význam | Definice zápisu | Příklad/ Příklad se zónou |
|---|---|---|---|
| date | datum | yyyy-MM-dd | 2012-04-13 / 2012-04-13-6:00 |
| time | čas | HH:mm:ss | 21:36:55 |
| dateTime | datum a čas | yyyy-MM-ddTHH:mm:ss | 2012-04-13T21:36:55 / 2012-04-13T21:36:55-6:00 |
| gYeart | rok | yyyy | 2012 |
| gMonth | měsíc | --MM | --08 / --08+1:00 |
| gDay | den | ---dd | ---08 |
| gYearMonth | rok a měsíc | yyyy-MM | 2012-04 / 2012-04+0:00 |
| gMonthDay | měsíc a den | --MM-dd | --04-13 / --04-13-11:00 |

Tabulka č.1 Přehled všech datových typů s datem nebo časem

Spinnery s daty a časy byly vybrány a použity díky modelu s datem a časem, který spinner podporuje. V objektu spinneru je možnost nastavit pomocí metody setEditor formát data a času ve vybraném formátu. Ukázka řešení je zobrazena v Příklad č.24, kde proměnná dateFormat je parametr konstruktoru třídy na vytvoření JPanelu s časem a datem ve nějakém z tvarů definice zápisu z Tabulka č.1.

```
SpinnerModel dateModel = new SpinnerDateModel(
    Calendar.getInstance().getTime(),//aktualni cas
    null,//minimalni hodnota (data a casu)
    null,//maximalni hodnota (data a casu)
    Calendar.YEAR
);
spinner = new JSpinner(dateModel);
this.add(spinner);
spinner.setEditor(new JSpinner.DateEditor(
    spinner,
    dateFormat
));
```

Příklad č.24 Vytvoření spinneru s datem a časem

Hodnoty dateFormat odpovídají definici zápisu z Tabulka č.1 každého datového typu s časem nebo datem. Text v komponentě je zobrazený ve výsledném formátu, kde místo znaků jsou čísla aktuálního času.

Vygenerovaný spinner například s datem, tudíž rok-měsíc-den, se ovládá buďto pomocí šipek na klávesnici nebo šipek v komponentě pomocí myši. Ovládá se vždycky ta část výrazu, kde je kurzor, takže když se klikne na měsíc, tak se ovládá měsíc. Navíc když se hodnota dostane na poslední možné políčko, nejnižší i nejvyšší, tak se hodnota ve vyšší úrovni, pokud existuje, taky změní o jedničku. V našem případě, kdybychom snižovali měsíc, tak po lednu následuje opět prosinec a rok klesne o jeden. To znamená, že všechny hodnoty ve spinneru jsou navzájem propojeny.

### Choice

Choice a jeho potomci představují speciální případ pro vygenerování komponent. Problém spočívá v tom, že komponenta ButtonGroup, jenž slučuje potomky choice do jedné skupiny, nedědí z JComponent, proto je jako komponenta panel s rámečkem a ButtonGroup je uložena do speciální proměnné určené výhradně pro choice.

Potomci choice se skládají ze dvou komponent, té hlavní co nesou sami a z přepínacího tlačítka, které je uloženo v objektu PrvekElementu u tagu choice, celé to je vyřešeno pomocí hash tabulky, která obsahuje jako klíč odkaz na potomky, které jsou neseřazené v listu potomků, tka jak to mají všechny tyto objekty, a hodnota je přepínací tlačítko. Pomocí listu se vybírají přepínací tlačítka z hash tabulky. List a hash tabulka jsou zobrazeny v Příklad č.25.

```
private List<PrvekSchema> potomci=
                new ArrayList<PrvekSchema>();
private Hashtable<PrvekSchema,JRadioButton>
                choiceComponent=
                new Hashtable<PrvekSchema,JRadioButton>();
```

Příklad č.25 Hash tabulka a list pro choice

**All**

Tam kde je potomek konstrukce all, musí být i tlačítka na posouvání v této konstrukci nahoru a dolů. Tyto tlačítka jsou generovány až při zobrazování komponent na obrazovku, na každé z těchto tlačítek je připojen posluchač, kterému jsou předány základní údaje o této komponentě, to je pořadí v konstrukci all, PrvekSchema s údaji dané komponenty, odkaz na vytvoření schématu, kvůli novému vykreslení.

Princip záměny dvou komponent v listu potomků u tagu all je poměrně jednoduchý, pokud se prvek bude posouvat v listu dolů, tak se uloží prvek, co je o úroveň níže, to znamená pozice prvku + 1, do proměnné a smaže se z listu, následně se smaže z listu daný prvek. Pak se tyto dva prvky uloží zpátky do listu, ale s opačnými pozicemi, první se musí začít nižší pozici. Celý tento kód je zobrazený v Příklad č.26.

```
PrvekSchema ps=schema.getPredek().getPotomci().get(pozice+1);
schema.getPredek().getPotomci().remove(pozice+1);
schema.getPredek().getPotomci().remove(pozice);
schema.getPredek().getPotomci().add(pozice, ps);
schema.getPredek().getPotomci().add(pozice+1, schema);
```

Příklad č.26 Ukázka posluchače pro posun komponenty v konstrukci all

To stejné platí i pro posluchače s vlastnosti posunout komponentu nahoru, jediný rozdíl je, že se nepřičítá jednička, ale odečítá.

**Počet výskytů jednotlivých tagu**

V případě nerovnosti čísel minOccurs a maxOccurs se musí zobrazit v aplikaci u těchto komponent tlačítko na přidání nebo odebrání těchto komponent. Vytvoření tlačítek je stejné jako u konstrukce all, tudíž až při vykreslování aplikace, kde každému tlačítku je přiřazený posluchač.

U odebrání se tento prvek pouze odebere z listu z PrvekSchema, který drží tyto své kopie. U přidání se prvek přidá na konec, ale musí se provést stejný způsob jako v kapitole 4.5 Vytvoření formuláře, takže se zavolá metoda projdi (Příklad č.23), která je rekurzivní, a proto vytvoří pro danou komponentu celý nový podstrom a ten na konci přidá do listu od PrvekSchema. Posledním krokem je po přidání i odebrání komponenty nutné tento strom vykreslit.

# 4.6 Vykreslení komponent

O vykreslení, to znamená změnu obsahu hlavního okna aplikace, se stará metoda tisk v třídě VytvorFormular.java zobrazena v Příklad č.27, kde parametr schema obsahuje údaje o tagu a list se svými prvky držící komponenty, parametr hlavni představuje komponentu předka, to je místo, kde se budou komponenty z listu prvků přidávat, maximum představuje počet potomků v předkovi, kvůli zobrazení tlačítek pro konstrukci all.

```
private void tisk(
    PrvekSchema schema,
    JComponent hlavni,
    int maximum
)
```

Příklad č.27 Deklarace metody tisk

Uvnitř této metody se zjišťuje, zda mají být vytvořeny tlačítka pro počty komponent a all, pro choice se sloučí přepínací tlačítka s potomky. Z parametru schema se projdou prvky daného tagu uložené v listu, uvnitř tohoto cyklu je přidání komponenty do předka, což je ve výsledku naše potřebné vykreslení, a následně se u každého prvku projdou cyklem jeho potomci, na které se použije rekurzivní metoda, tento cyklus je zobrazený v Příklad č.28.

```
pocet=schema.getPredek().getPocet();
gbc.gridy=pocet;
hlavni.add(prvkyElementu.getPrvek(),gbc);
schema.getPredek().setPocet(pocet+1);

Iterator<PrvekSchema> it2=
    prvkyElementu.getPotomci().iterator();
while(it2.hasNext())
{
    tisk(
        it2.next(),
        prvkyElementu.getPrvek(),
        prvkyElementu.getPotomci().size()
    );
}
```

Příklad č.28 Zobrazení komponenty a rekurzivní volání metody tisk

Pro vykreslení komponent na správné pozice se používá GridBagLayout, který se nastavuje při vytváření všem panelovým komponentám. V Tabulka č.2 je ukázané vertikální rozestavění komponent. Pro element jde využít všechny políčka, pro panel se použije pouze gridx s 2.

| gridx=0 | gridx=1 | gridx=2 | gridx=4 | gridx=5 | gridx=6 | gridx=7 | gridx=8 |
|---|---|---|---|---|---|---|---|
| Přepínací tlačítko | Název | Komponenta | Povinnost * | Nahoru | Dolů | Odebrat | Přidat |

Tabulka č.2 Vertikální rozložení komponenty

Horizontální hodnoty rozmístění má každá komponenta pro své potomky vlastní, tudíž od nuly. Z toho plyne, že počítadlo je umístěno v PrvkyElementu. Každé další zobrazená komponenta přičte k počítadlu u předka jedničku, ať následující komponenta jí nepřepíše a posune se o řádek níže.

# 4.7 Testování

Při implementaci aplikace bylo samozřejmostí provádět testování, bez něhož by ani nebylo možné program vytvořit. Testování bylo prováděno téměř po každé menší naimplementované části, ať už v průběhu vytvoření nové metody, třídy, pomocně třídy, atp.

Při testování byly vytvořeny a použity různé typy návrhu dokumentů s různými datovými typy, tagy a komponentami. Postupem času byly otestovány všechny komponenty a základní datové typy. Ve finální verzi byly znovu odzkoušeny všechny tyto dokumenty pro ověření správného chování aplikace. Aplikace byla testována pouze na platformě Windows XP Profesionál 32-bit, ale v tom by neměl být problém, jelikož Java je multiplatformní.

# 5 Závěr

Téma této bakalářské práce bylo dynamické vytváření formulářů z XSD dokumentů. Celá aplikace měla být vytvořená v programovacím jazyce Java SE pomocí její knihovny Swing, což bylo mým hlavním důvodem, proč jsem si tuto práci vybral, neboť jazyk Java a knihovnu Swing ovládám. Na začátku práce byl pro mě soubor XSD velkou neznámou, a tak jsem byl nucen tento problém nastudoval.

Cílem práce tedy bylo vytvořit pomocí Swingu formulář odpovídající vloženému XSD souboru a následné uložení do XML. Výsledkem se stala desktopová aplikace, odpovídající zadání práce. Po prostudování jsem měl ihned představu, jak aplikace bude vypadat, i myšlenku, jak celou aplikaci sestavit, jež jsem se po celou dobu vývoje držel, neboť tento postup byl vhodný pro výslednou aplikaci. Výsledná aplikace dokáže zpracovat a zobrazit vysoké množství různých typů XSD dokumentů s různými tagy, což jsem ověřil testováním v průběhu implementace i na konci vytvořené finální verze.

Při implementaci aplikace jsem občas nalezl v nových studijních materiálech dosti nepoužívané elementy a atributy XSD dokumentů, o kterých téměř ve všech ostatních studijních materiálech nebyla ani zmínka. Většina těchto mě utajených vlastností ani nehraje žádnou roli při vygenerování formuláře, a když ano, tak jsem se je snažil doimplementovat, ale ne u všech se to povedlo, ale všechny tyto vlastnosti jsou zanedbatelné a výsledný formulář by to nezměnilo. Na začátku práce byly po dohodě s vedoucím vypuštěny dvě věci. První je vytvoření komponent ve formuláři pro smíšené typy, druhou věcí je importování, do vybraného XSD souboru, dalších XSD soubory. Tyto dva hlavní nedostatky by bylo vhodné do budoucna dořešit a doplnit, ty zanedbatelné vlastnosti by se mohly taky dodělat, pro úplnou dokonalost celé aplikace a zvýšení jejího hodnocení.

Na internetu jsem našel pár podobných aplikací a většina z nich byla placená a nestály zrovna málo. Tento typ aplikací využívají především firmy pracující hodně s různými typy XML dokumentů. Nevím, zda nějaká taková firma nebo někdo bude používat v budoucnu mnou vytvořenou aplikaci, ale kdyby ano, tak bych byl za to rád a potěšilo by mě to. Já osobně nemám možnost, kde bych tuto aplikaci použil, ale když jsou XML dokumenty hojně využívány, tak třeba v budoucnu se bude hodit a určitě s radostí jí použiju.

# 6 Literatura


1. **BRAY, Tim, a další, a další.** Extensible Markup Language (XML) 1.0 (Fifth Edition). *W3C*. [Online] 26. Prosinec 2008. [Citace: 2012. Březen 15.] http://www.w3.org/TR/xml/.

2. **HEROUT, Pavel.** *Java a XML*. České Budějovice : Kopp, 2007. ISBN 978-80-7232-307-4.

3. **ST. LAURENT, Simon.** *Tvorba internetových aplikací v XML*. [překl.] Jan Škvařil. Brno : Computer Press, 1999. stránky 53-85. ISBN 80-7226-170-3.

4. **FALLSIDE, David C. a WALMSLEY, Priscilla.** XML Schema Part 0: Primer Second Edition. *W3C*. [Online] 28 October 2004. říjen 2004. [Citace: 2012. březen 15.] http://www.w3.org/TR/xmlschema-0/.

5. **SKONNARD, Aaron a GUDGIN, Martin.** *XML: pohotová referenční příručka*. [překl.] Lucie Rút BITTNEROVÁ. Praha : Grada Publishing, 2006. stránky 151-214. ISBN 80-247-0972-4.

6. **KOSEK, Jiří.** XML schémata. *Téměř vše o WWW*. [Online] 18. srpen 2005. [Citace: 21. březen 2012.] http://www.kosek.cz.

7. **HATINA, Petr a JELÍNEK, Lukáš.** Java seriál. *linuxsoft*. [Online] [Citace: 2012. březen 18.] http://www.linuxsoft.cz/article_list.php?id_kategory=192.

8. **HEROUT, Pavel.** *Java - grafické uživatelské prostředí a čeština*. České Budějovice : Kopp, 2007. ISBN 978-80-7232-328-9.

9. **SOJKA, Eduard.** Uživatelské rozhraní - úvod. *Uživatelské rozhraní*. [Online] 2011. [Citace: 27. březen 2012.] http://mrl.cs.vsb.cz/people/sojka/uro/uvod.pdf.

10. **NetBeans.** [Online] http://netbeans.org/.

# Obsah přiloženého CD

| | |
|---|---|
| /Aplikace | vygenerovaný jar soubor |
| /Javadoc | vygenerovaná programátorská dokumentace |
| /Schema | přiložené ukázkové XSD dokumenty |
| /Dokumentace | text bakalářské práce |
| /Zdrojovy_kod | zdrojové kódy aplikace z vývojového prostředí NetBeans |